



MOTOROLA

Semiconductors

8-BIT MICROPROCESSING UNIT (MPU)

The MC6800 is a monolithic 8-bit microprocessor forming the central control function for Motorola's M6800 family. Compatible with TTL, the MC6800, as with all M6800 system parts, requires only one +5.0-volt power supply, and no external TTL devices for bus interface.

The MC6800 is capable of addressing 65K bytes of memory with its 16-bit address lines. The 8-bit data bus is bidirectional as well as 3-state, making direct memory addressing and multiprocessing applications realizable.

- Eight-Bit Parallel Processing
- Bidirectional Data Bus
- Sixteen-Bit Address Bus — 65K Bytes of Addressing
- 72 Instructions — Variable Length
- Seven Addressing Modes — Direct, Relative, Immediate, Indexed, Extended, Implied and Accumulator
- Variable Length Stack
- Vectored Restart
- Maskable Interrupt Vector
- Separate Non-Maskable Interrupt — Internal Registers Saved in Stack
- Six Internal Registers — Two Accumulators, Index Register, Program Counter, Stack Pointer and Condition Code Register
- Direct Memory Addressing (DMA) and Multiple Processor Capability
- Simplified Clocking Characteristics
- Clock Rates as High as 2.0 MHz
- Simple Bus Interface Without TTL
- Halt and Single Instruction Execution Capability

ORDERING INFORMATION

Speed	Device	Temperature Range
1.0 MHz MIL-STD-883B MIL-STD-883C	MC6800P, L	0 to 70°C
	MC6800CP, CL	-40 to +85°C
	MC6800BQCS	-55 to +125°C
	MC6800CQCS	
1.5 MHz	MC68A00P, L	0 to +70°C
	MC68A00CP, CL	-40 to +85°C
2.0 MHz	MC68B00P, L	0 to +70°C

MC6800

(1.0 MHz)

MC68A00

(1.5 MHz)

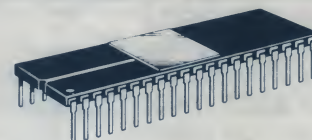
MC68B00

(2.0 MHz)

MOS

(N-CHANNEL, SILICON-GATE,
DEPLETION LOAD)

MICROPROCESSOR



L SUFFIX
CERAMIC PACKAGE
CASE 715



P SUFFIX
PLASTIC PACKAGE
CASE 711

PIN ASSIGNMENT

1	V _{SS}	Reset	40
2	Halt	TSC	39
3	φ1	N.C.	38
4	IRQ	φ2	37
5	V _{MA}	DBE	36
6	NMI	N.C.	35
7	BA	R/W	34
8	V _{CC}	D0	33
9	A0	D1	32
10	A1	D2	31
11	A2	D3	30
12	A3	D4	29
13	A4	D5	28
14	A5	D6	27
15	A6	D7	26
16	A7	A15	25
17	A8	A14	24
18	A9	A13	23
19	A10	A12	22
20	A11	V _{SS}	21

TABLE 1 — MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	-0.3 to +7.0	Vdc
Input Voltage	V_{in}	-0.3 to +7.0	Vdc
Operating Temperature Range— T_L to T_H MC6800, MC68A00, MC68B00 MC6800C, MC68A00C MC6800BQCS, MC6800CQCS	T_A	0 to +70 -40 to +85 -55 to +125	$^{\circ}\text{C}$
Storage Temperature Range	T_{stg}	-55 to +150	$^{\circ}\text{C}$
Thermal Resistance Plastic Package Ceramic Package	θ_{JA}	70 50	$^{\circ}\text{C/W}$

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

TABLE 2 — ELECTRICAL CHARACTERISTICS ($V_{CC} = 5.0 \text{ V}$, $\pm 5\%$, $V_{SS} = 0$, $T_A = T_L$ to T_H unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage Logic $\phi 1, \phi 2$	V_{IH} V_{IHC}	$V_{SS} + 2.0$ $V_{CC} - 0.6$	— —	V_{CC} $V_{CC} + 0.3$	Vdc
Input Low Voltage Logic $\phi 1, \phi 2$	V_{IL} V_{ILC}	$V_{SS} - 0.3$ $V_{SS} - 0.3$	— —	$V_{SS} + 0.8$ $V_{SS} + 0.4$	Vdc
Input Leakage Current ($V_{in} = 0$ to 5.25 V , $V_{CC} = \text{max}$) ($V_{in} = 0$ to 5.25 V , $V_{CC} = 0.0 \text{ V}$)	I_{in} Logic* $\phi 1, \phi 2$	— —	1.0 —	2.5 100	μAdc
Three-State (Off State) Input Current ($V_{in} = 0.4$ to 2.4 V , $V_{CC} = \text{max}$) D0-D7 A0-A15, R/W	I_{TSI}	— —	2.0 —	10 100	μAdc
Output High Voltage ($I_{Load} = -205 \mu\text{Adc}$, $V_{CC} = \text{min}$) ($I_{Load} = -145 \mu\text{Adc}$, $V_{CC} = \text{min}$) ($I_{Load} = -100 \mu\text{Adc}$, $V_{CC} = \text{min}$) D0-D7 A0-A15, R/W, VMA BA	V_{OH}	$V_{SS} + 2.4$ $V_{SS} + 2.4$ $V_{SS} + 2.4$	— — —	— — —	Vdc
Output Low Voltage ($I_{Load} = 1.6 \text{ mAdc}$, $V_{CC} = \text{min}$)	V_{OL}	—	—	$V_{SS} + 0.4$	Vdc
Power Dissipation	P_D	—	0.5	1.0	W
Capacitance ($V_{in} = 0$, $T_A = 25^{\circ}\text{C}$, $f = 1.0 \text{ MHz}$) $\phi 1$ $\phi 2$ D0-D7 Logic Inputs A0-A15, R/W, VMA	C_{in} C_{out}	— — — — —	25 45 10 6.5	35 70 12.5 10	pF

TABLE 3 — CLOCK TIMING ($V_{CC} = 5.0 \text{ V}$, $\pm 5\%$, $V_{SS} = 0$, $T_A = T_L$ to T_H unless otherwise noted)

Characteristics	Symbol	Min	Typ	Max	Unit
Frequency of Operation MC6800 MC68A00 MC68B00	f	0.1 0.1 0.1	— — —	1.0 1.5 2.0	MHz
Cycle Time (Figure 1) MC6800 MC68A00 MC68B00	t_{cyc}	1.000 0.666 0.500	— — —	10 10 10	μs
Clock Pulse Width (Measured at $V_{CC} - 0.6 \text{ V}$) $\phi 1, \phi 2$ — MC6800 $\phi 1, \phi 2$ — MC68A00 $\phi 1, \phi 2$ — MC68B00	$PW_{\phi H}$	400 230 180	— — —	9500 9500 9500	ns
Total $\phi 1$ and $\phi 2$ Up Time MC6800 MC68A00 MC68B00	t_{ut}	900 600 440	— — —	— — —	ns
Rise and Fall Times (Measured between $V_{SS} + 0.4$ and $V_{CC} - 0.6$)	$t_{\phi r}, t_{\phi f}$	—	—	100	ns
Delay Time or Clock Separation (Figure 1) (Measured at $V_{OV} = V_{SS} + 0.6 \text{ V}$ @ $t_r = t_f \leq 100 \text{ ns}$) (Measured at $V_{OV} = V_{SS} + 1.0 \text{ V}$ @ $t_r = t_f \leq 35 \text{ ns}$)	t_d	0 0	— —	9100 9100	ns



TABLE 4 – READ/WRITE TIMING (Reference Figures 2 through 6)

Characteristic	Symbol	MC6800			MC68A00			MC68B00			Unit
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Address Delay C = 90 pF C = 30 pF	t_{AD}	—	—	270 250	—	—	180 165	—	—	150 135	ns
Peripheral Read Access Time $t_{ac} = t_{ut} - (t_{AD} + t_{DSR})$	t_{acc}	—	—	530	—	—	360	—	—	250	ns
Data Setup Time (Read)	t_{DSR}	100	—	—	60	—	—	40	—	—	ns
Input Data Hold Time	t_H	10	—	—	10	—	—	10	—	—	ns
Output Data Hold Time	t_H	10	25	—	10	25	—	10	25	—	ns
Address Hold Time (Address, R/W, VMA)	t_{AH}	30	50	—	30	50	—	30	50	—	ns
Enable High Time for DBE Input	t_{EH}	450	—	—	280	—	—	220	—	—	ns
Data Delay Time (Write)	t_{DDW}	—	—	225	—	—	200	—	—	160	ns
Processor Controls											
Processor Control Setup Time	t_{PCS}	200	—	—	140	—	—	110	—	—	ns
Processor Control Rise and Fall Time	t_{PCr}, t_{PCf}	—	—	100	—	—	100	—	—	100	ns
Bus Available Delay	t_{BA}	—	—	250	—	—	165	—	—	135	ns
Three-State Delay	t_{TSD}	—	—	270	—	—	270	—	—	220	ns
Data Bus Enable Down Time	t_{DBE}	150	—	—	120	—	—	75	—	—	ns
During ϕ_1 Up Time											
Data Bus Enable Rise and Fall Times	t_{DBEr}, t_{DBEf}	—	—	25	—	—	25	—	—	25	ns

FIGURE 1 – CLOCK TIMING WAVEFORM

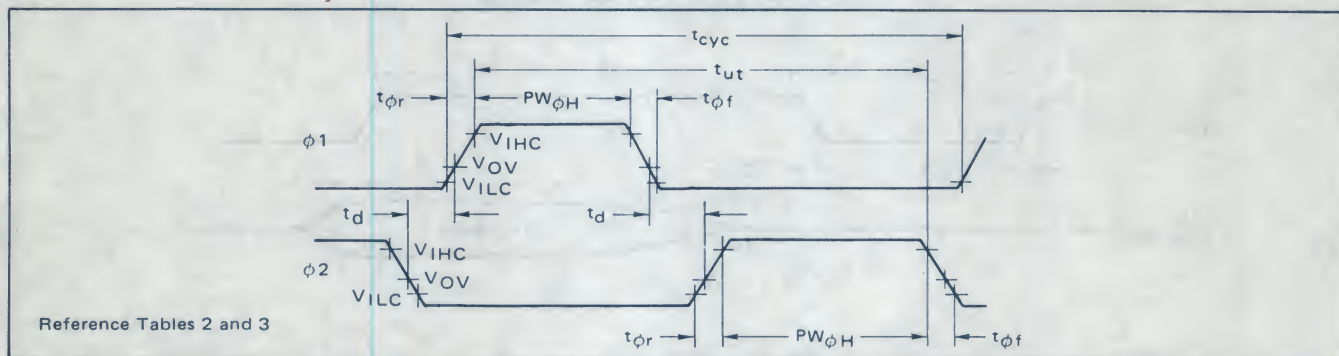


FIGURE 2 – READ DATA FROM MEMORY OR PERIPHERALS

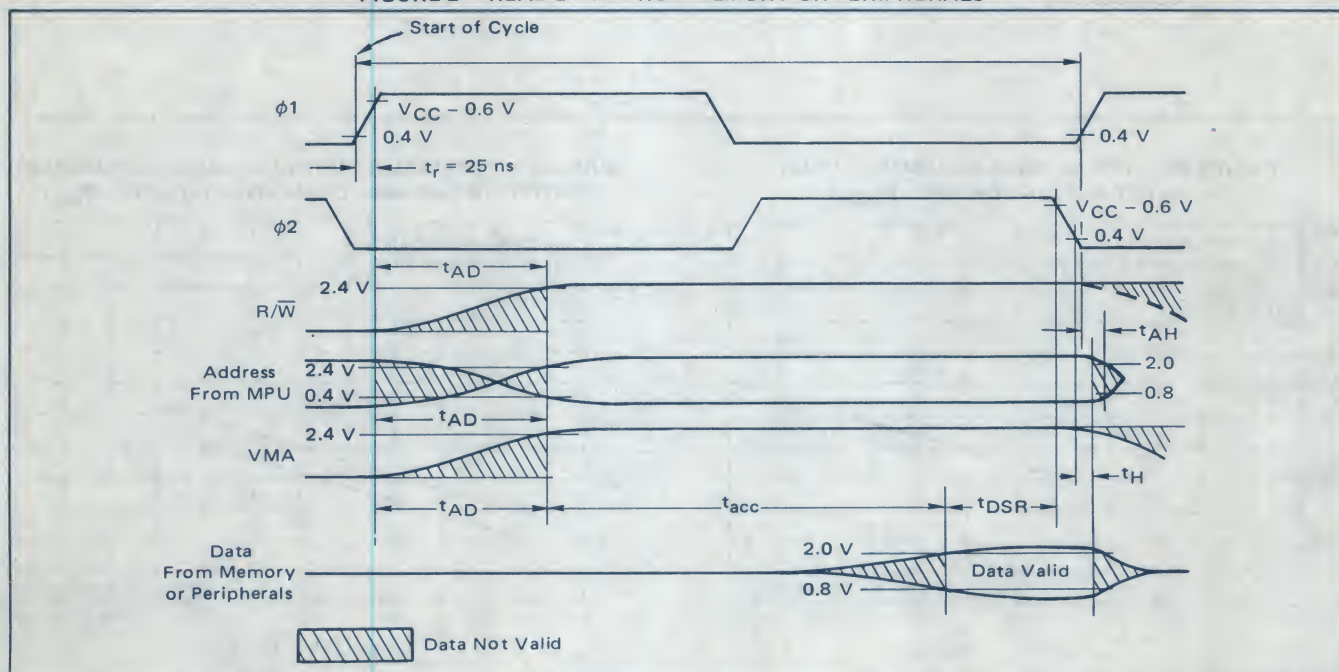


FIGURE 3 – WRITE IN MEMORY OR PERIPHERALS

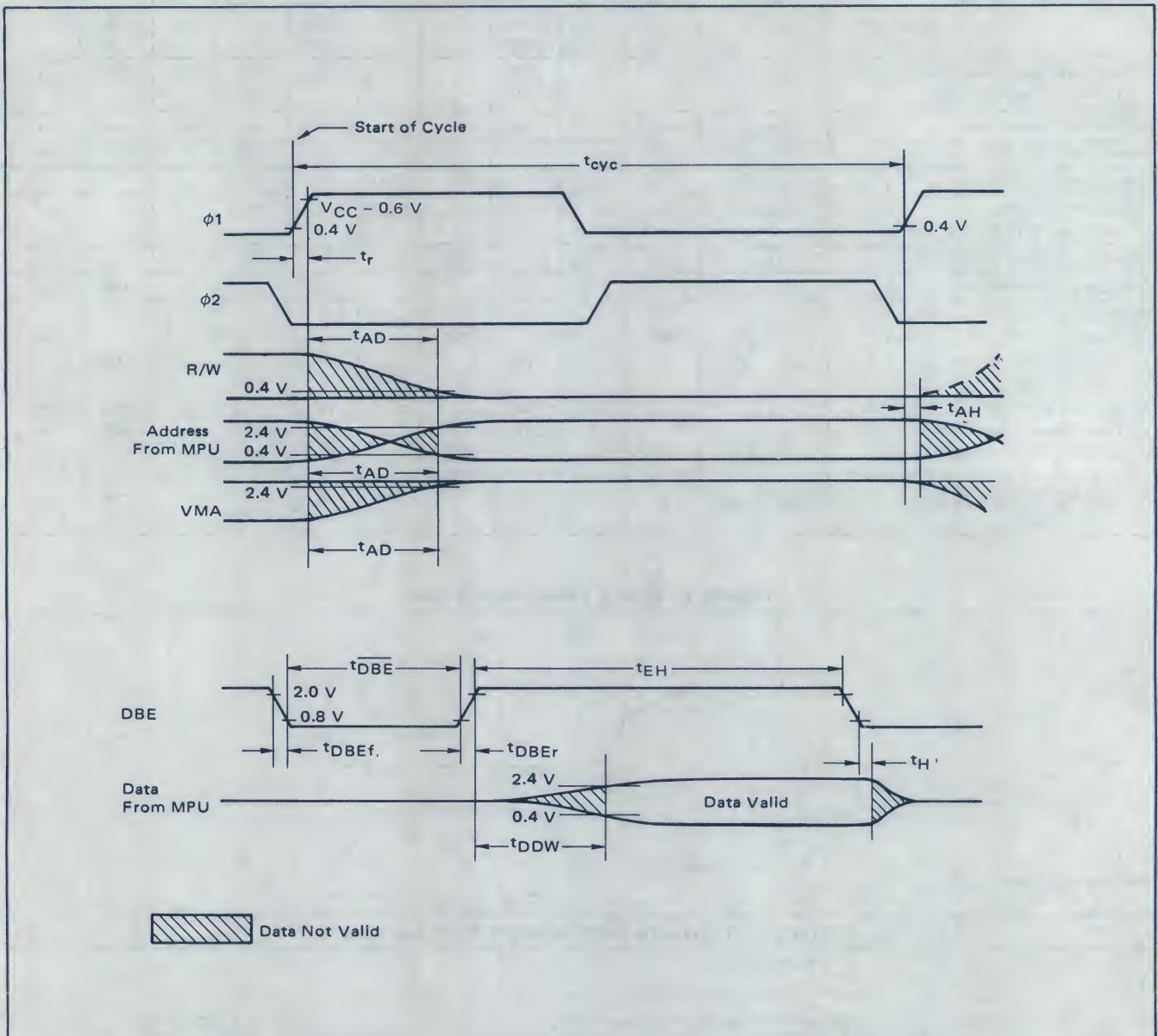
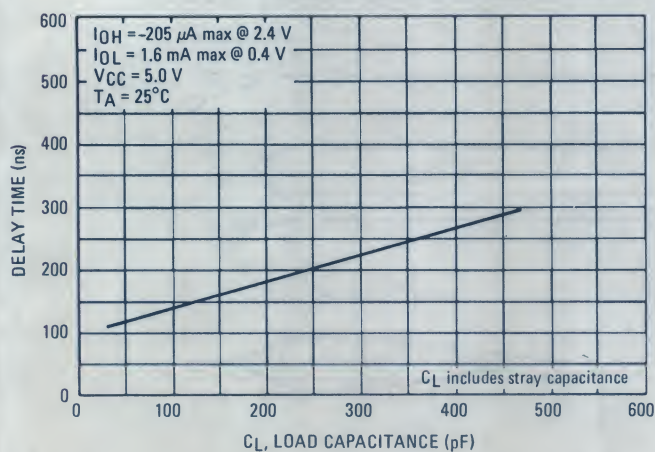
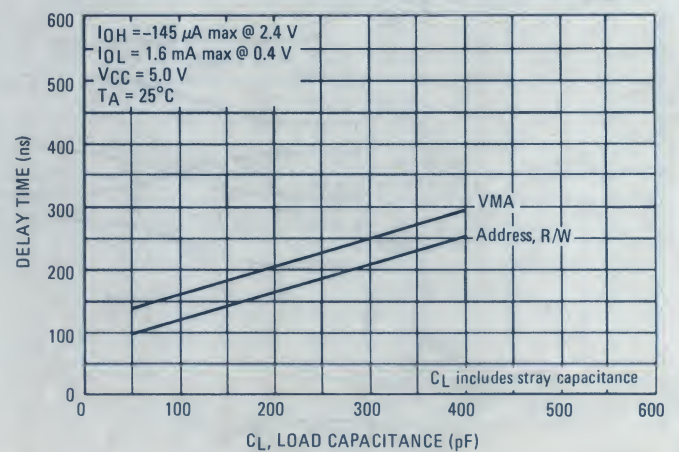
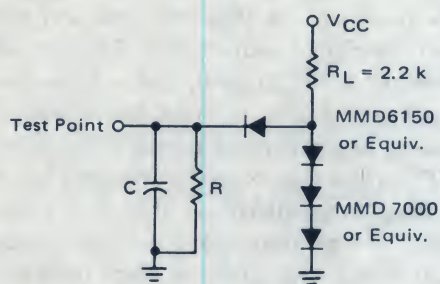
FIGURE 4 – TYPICAL DATA BUS OUTPUT DELAY versus CAPACITIVE LOADING (T_{DDW})FIGURE 5 – TYPICAL READ/WRITE, VMA, AND ADDRESS OUTPUT DELAY versus CAPACITIVE LOADING (T_{AD})

FIGURE 6 – BUS TIMING TEST LOADS



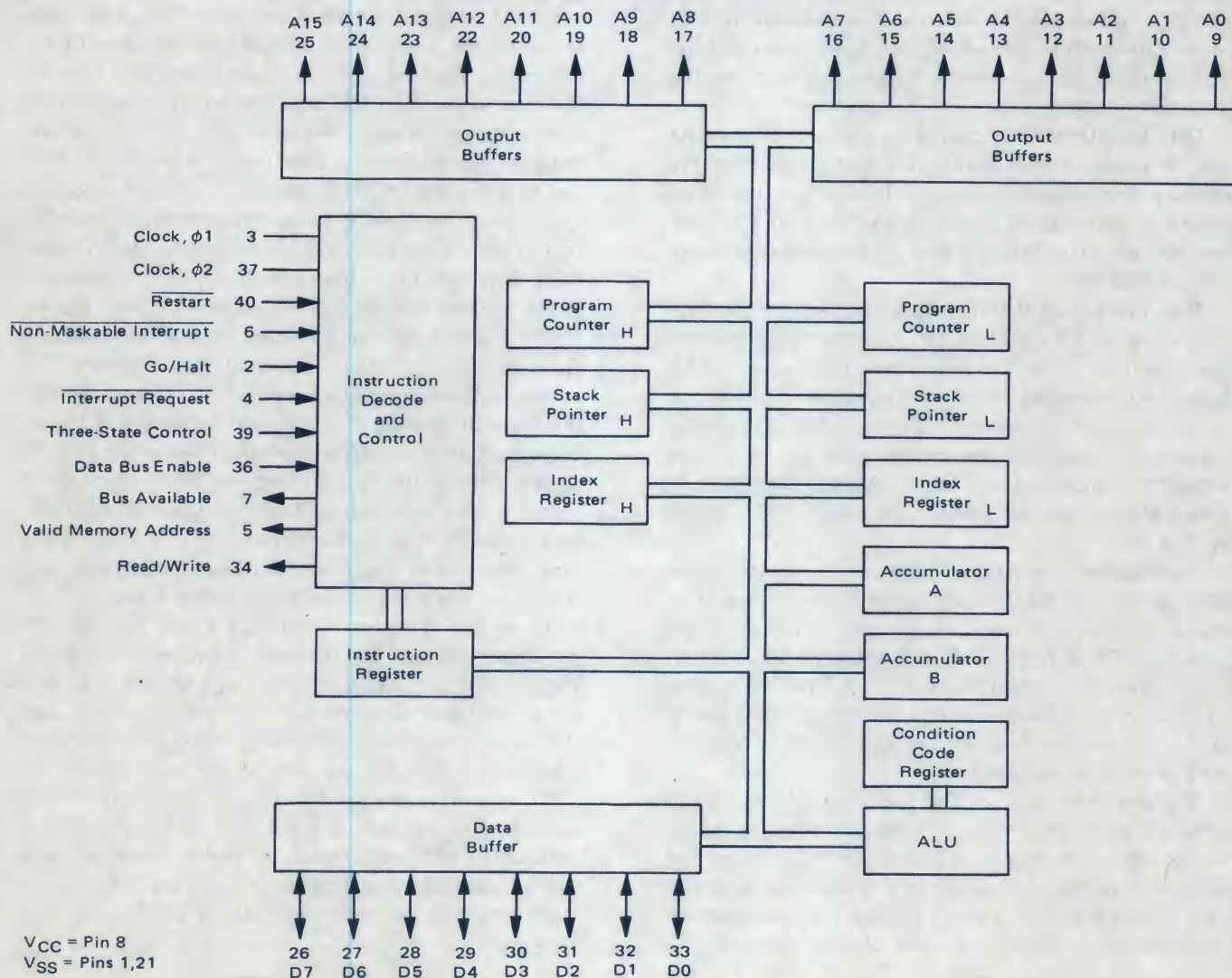
- $C = 130 \text{ pF}$ for D0-D7, E
 $= 90 \text{ pF}$ for A0-A15, R/W, and VMA
 (Except t_{AD2})
 $= 30 \text{ pF}$ for A0-A15, R/W, and VMA
 (t_{AD2} only)
 $= 30 \text{ pF}$ for BA
 $R = 11.7 \text{ k}\Omega$ for D0-D7
 $= 16.5 \text{ k}\Omega$ for A0-A15, R/W, and VMA
 $= 24 \text{ k}\Omega$ for BA

TEST CONDITIONS

The dynamic test load for the Data Bus is 130 pF and one standard TTL load as shown. The Address, R/W, and VMA outputs are tested under two conditions to allow optimum operation in both buffered and unbuffered systems. The resistor (R) is chosen to insure specified load currents during V_{OH} measurement.

Notice that the Data Bus lines, the Address lines, the Interrupt Request line, and the DBE line are all specified and tested to guarantee 0.4 V of dynamic noise immunity at both "1" and "0" logic levels.

FIGURE 7 – EXPANDED BLOCK DIAGRAM



MPU SIGNAL DESCRIPTION

Proper operation of the MPU requires that certain control and timing signals be provided to accomplish specific functions and that other signal lines be monitored to determine the state of the processor.

Clocks Phase One and Phase Two ($\phi 1, \phi 2$) — Two pins are used for a two-phase non-overlapping clock that runs at the V_{CC} voltage level.

Figure 1 shows the microprocessor clocks, and Table 3 shows the static and dynamic clock specifications. The high level is specified at V_{IHc} and the low level is specified at V_{ILc} . The allowable clock frequency is specified by f (frequency). The minimum $\phi 1$ and $\phi 2$ high level pulse widths are specified by $PW_{\phi H}$ (pulse width high time). To guarantee the required access time for the peripherals, the clock up time, t_{ut} , is specified. Clock separation, t_d , is measured at a maximum voltage of V_{OV} (overlap voltage). This allows for a multitude of clock variations at the system frequency rate.

Address Bus (A0–A15) — Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 90 pF. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications. Putting TSC in its high state forces the Address bus to go into the three-state mode.

Data Bus (D0–D7) — Eight pins are used for the data bus. It is bidirectional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130 pF. Data Bus is placed in the three-state mode when DBE is low.

Data Bus Enable (DBE) — This input is the three-state control signal for the MPU data bus and will enable the bus drivers when in the high state. This input is TTL compatible; however in normal operation, it would be driven by the phase two clock. During an MPU read cycle, the data bus drivers will be disabled internally. When it is desired that another device control the data bus such as in Direct Memory Access (DMA) applications, DBE should be held low.

If additional data setup or hold time is required on an MPU write, the DBE down time can be decreased as shown in Figure 3 ($DBE \neq \phi 2$). The minimum down time for DBE is t_{DBE} as shown and must occur within $\phi 1$ up time. The minimum delay from the trailing edge of DBE to the trailing edge of $\phi 1$ is t_{DBED} . By skewing DBE with respect to E in this manner, data setup or hold time can be increased.

Bus Available (BA) — The Bus Available signal will normally be in the low state; when activated, it will go to the high state indicating that the microprocessor has stopped and that the address bus is available. This will occur if the Halt line is in the low state or the processor is in the WAIT state as a result of the execution of a

WAIT instruction. At such time, all three-state output drivers will go to their off state and other outputs to their normally inactive level. The processor is removed from the WAIT state by the occurrence of a maskable (mask bit $I = 0$) or nonmaskable interrupt. This output is capable of driving one standard TTL load and 30 pF. If TSC is in the high state, Bus Available will be low.

Read/Write (R/\bar{W}) — This TTL compatible output signals the peripherals and memory devices whether the MPU is in a Read (high) or Write (low) state. The normal standby state of this signal is Read (high). Three-State Control going high will turn Read/Write to the off (high impedance) state. Also, when the processor is halted, it will be in the off state. This output is capable of driving one standard TTL load and 90 pF.

Reset — The Reset input is used to reset and start the MPU from a power down condition resulting from a power failure or initial start-up of the processor. This input can also be used to reinitialize the machine at any time after start-up.

If a high level is detected in this input, this will signal the MPU to begin the reset sequence. During the reset sequence, the contents of the last two locations (FFFE, FFFF) in memory will be loaded into the Program Counter to point to the beginning of the reset routine. During the reset routine, the interrupt mask bit is set and must be cleared under program control before the MPU can be interrupted by IRQ. While Reset is low (assuming a minimum of 8 clock cycles have occurred) the MPU output signals will be in the following states: VMA = low, BA = low, Data Bus = high impedance, R/\bar{W} = high (read state), and the Address Bus will contain the reset address FFFE. Figure 8 illustrates a power up sequence using the Reset control line. After the power supply reaches 4.75 V a minimum of eight clock cycles are required for the processor to stabilize in preparation for restarting. During these eight cycles, VMA will be in an indeterminate state so any devices that are enabled by VMA which could accept a false write during this time (such as a battery-backed RAM) must be disabled until VMA is forced low after eight cycles. Reset can go high asynchronously with the system clock any time after the eighth cycle.

Reset timing is shown in Figure 8 and Table 4. The maximum rise and fall transition times are specified by t_{PCr} and t_{PCf} . If Reset is high at t_{PCS} (processor control setup time) as shown in Figure 8 in any given cycle, then the restart sequence will begin on the next cycle as shown. The Reset control line may also be used to reinitialize the MPU system at any time during its operation. This is accomplished by pulsing Reset low for the duration of a minimum of three complete $\phi 2$ cycles. The Reset pulse can be completely asynchronous with the MPU system clock and will be recognized during $\phi 2$ if setup time t_{PCS} is met.



FIGURE 8 — RESET TIMING

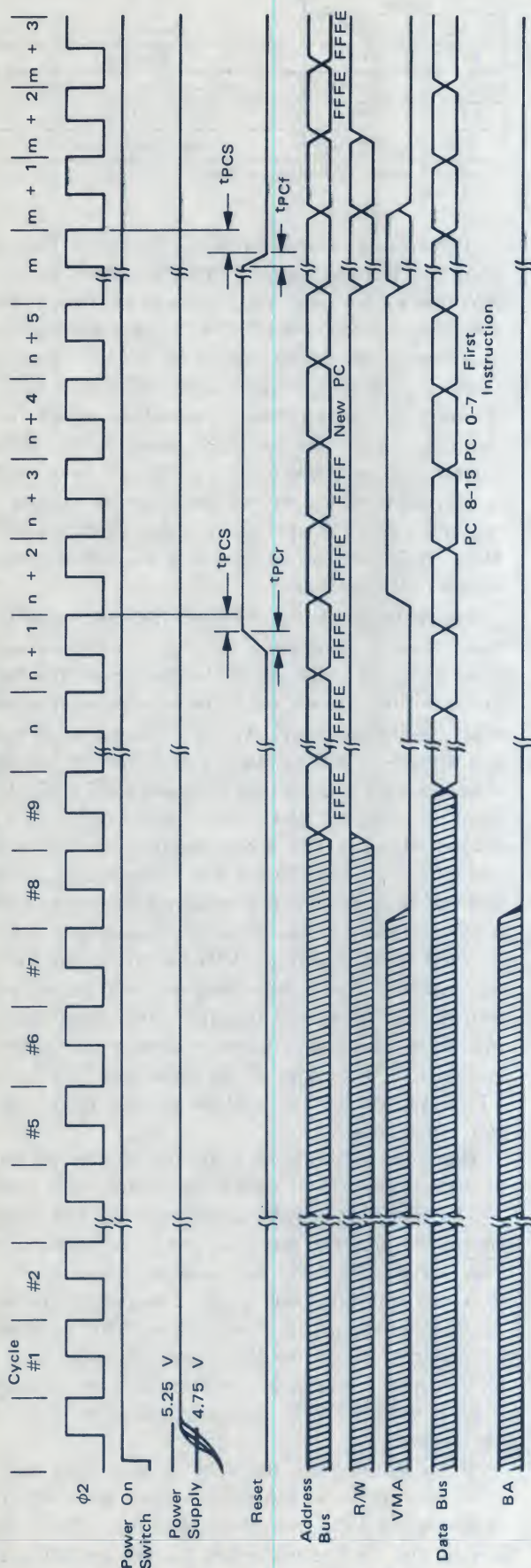
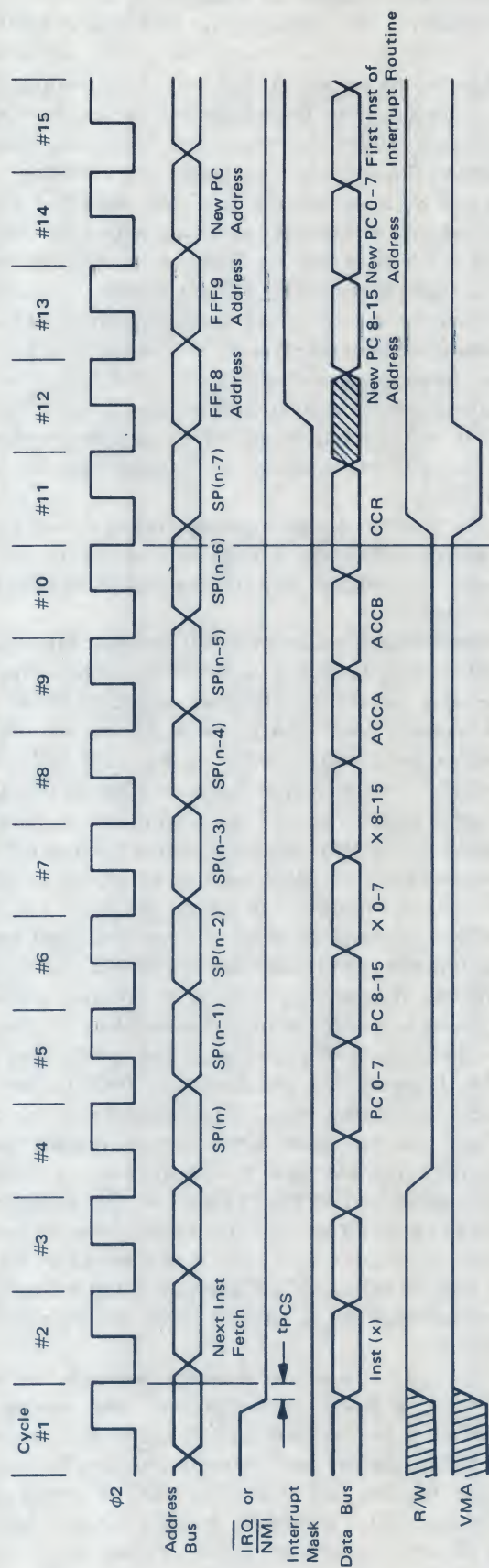


FIGURE 9 — INTERRUPT TIMING



Interrupt Request (IRQ) — This level sensitive input requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set, the machine will begin an interrupt sequence. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFF8 and FFF9. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory. Interrupt timing is shown in Figure 9.

The $\overline{\text{Halt}}$ line must be in the high state for interrupts to be serviced. Interrupts will be latched internally while $\overline{\text{Halt}}$ is low.

The $\overline{\text{IRQ}}$ has a high impedance pullup device internal to the chip; however a 3 k Ω external resistor to V_{CC} should be used for wire-OR and optimum control of interrupts.

Non-Maskable Interrupt (NMI) and Wait for Interrupt (WAI) — The MC6800 is capable of handling two types of interrupts: maskable ($\overline{\text{IRQ}}$) as described earlier, and non-maskable ($\overline{\text{NMI}}$). $\overline{\text{IRQ}}$ is maskable by the interrupt mask in the condition code register while $\overline{\text{NMI}}$ is not maskable. The handling of these interrupts by the MPU is the same except that each has its own vector address. The behavior of the MPU when interrupted is shown in Figure 9 which details the MPU response to an interrupt while the MPU is executing the control program. The interrupt shown could be either $\overline{\text{IRQ}}$ or $\overline{\text{NMI}}$ and can be asynchronous with respect to $\phi 2$. The interrupt is shown going low at time t_{PCS} in cycle #1 which precedes the first cycle of an instruction (OP code fetch). This instruction is not executed but instead the Program Counter (PC), Index Register (IX), Accumulators (ACCX), and the Condition Code Register (CCR) are pushed onto the stack.

The Interrupt Mask bit is set to prevent further interrupts. The address of the interrupt service routine is then fetched from FFFC, FFFD for an $\overline{\text{NMI}}$ interrupt and from FFF8, FFF9 for an $\overline{\text{IRQ}}$ interrupt. Upon completion of the interrupt service routine, the execution of RTI will pull the PC, IX, ACCX, and CCR off of the stack; the Interrupt Mask bit is restored to its condition prior to Interrupts.

Figure 11 is a similar interrupt sequence, except in this case, a WAIT instruction has been executed in preparation for the interrupt. This technique speeds up the MPU's response to the interrupt because the stacking of the PC, IX, ACCX, and the CCR is already done. While the MPU is waiting for the interrupt, Bus Available will go high indicating the following states of the control lines: VMA is low, and the Address Bus, $\overline{\text{R/W}}$ and Data Bus are all in the high impedance state. After the interrupt occurs, it is serviced as previously described.

TABLE 1 — MEMORY MAP FOR INTERRUPT VECTORS

Vector		Description
MS	LS	
FFFE	FFFF	Restart
FFFC	FFFD	Non-maskable Interrupt
FFFA	FFFB	Software Interrupt
FFF8	FFF9	Interrupt Request

Refer to Figure 11 for program flow for Interrupts.

Three State Control (TSC) — When the Three-State Control (TSC) line is a logic "1", the Address Bus and the $\overline{\text{R/W}}$ line are placed in a high impedance state. VMA and BA are forced low when TSC = "1" to prevent false reads or writes on any device enabled by VMA. It is necessary to delay program execution while TSC is held high. This is done by insuring that no transitions of $\phi 1$ (or $\phi 2$) occur during this period. (Logic levels of the clocks are irrelevant so long as they do not change.) Since the MPU is a dynamic device, the $\phi 1$ clock can be stopped for a maximum time $PW_{\phi H}$ without destroying data within the MPU. TSC then can be used in a short Direct Memory Access (DMA) application.

Figure 12 shows the effect of TSC on the MPU. TSC must have its transitions at t_{TSE} (three-state enable) while holding $\phi 1$ high and $\phi 2$ low as shown. The Address Bus and $\overline{\text{R/W}}$ line will reach the high impedance state at t_{TSD} (three-state delay), with VMA being forced low. In this example, the Data Bus is also in the high impedance state while $\phi 2$ is being held low since $\overline{\text{DBE}} = \phi 2$. At this point in time, a DMA transfer could occur on cycles #3 and #4. When TSC is returned low, the MPU Address and $\overline{\text{R/W}}$ lines return to the bus. Because it is too late in cycle #5 to access memory, this cycle is dead and used for synchronization. Program execution resumes in cycle #6.

Valid Memory Address (VMA) — This output indicates to peripheral devices that there is a valid address on the address bus. In normal operation, this signal should be utilized for enabling peripheral interfaces such as the PIA and ACIA. This signal is not three-state. One standard TTL load and 90 pF may be directly driven by this active high signal.

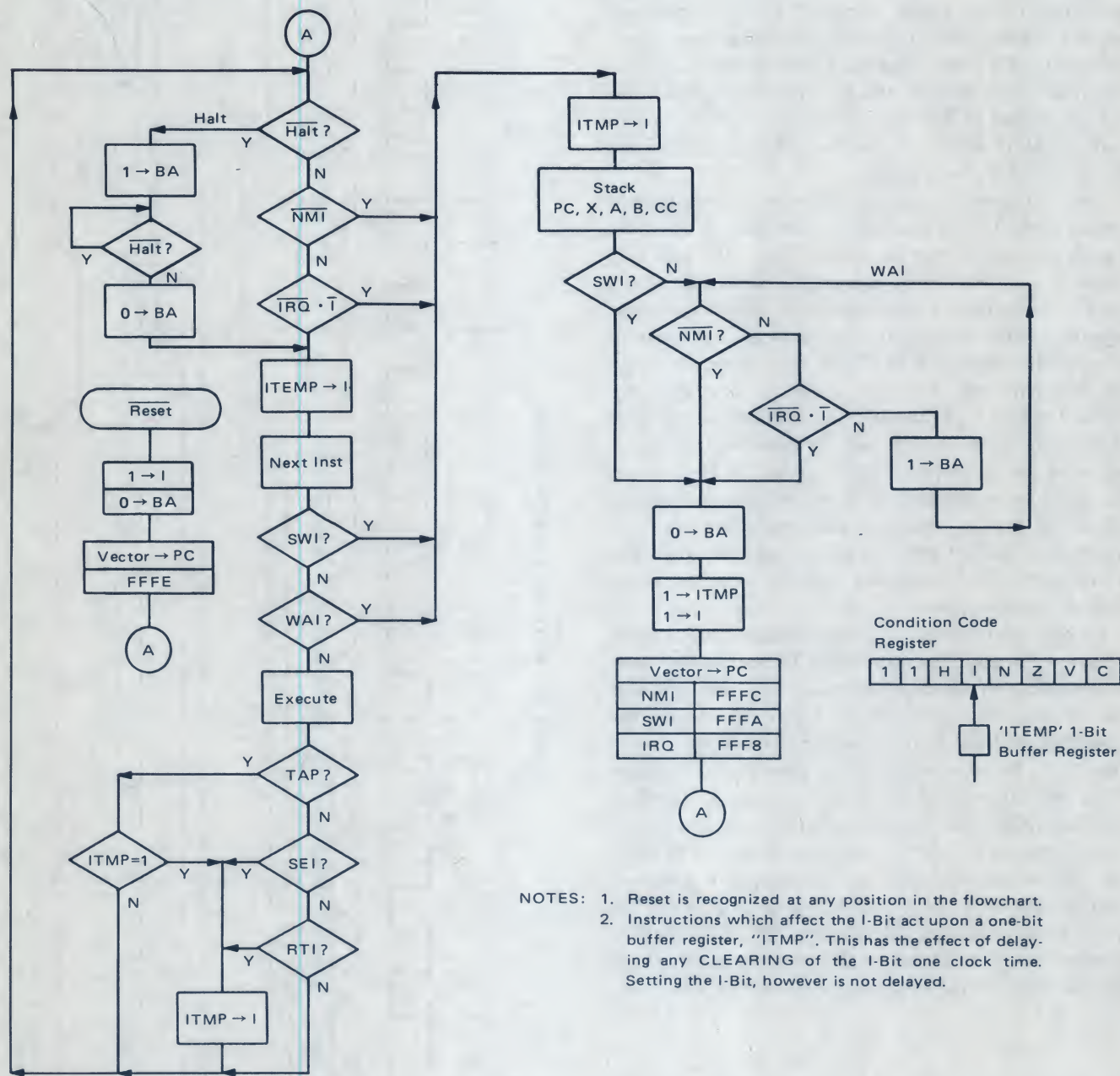
Halt — When this input is in the low state, all activity in the machine will be halted. This input is level sensitive.

The $\overline{\text{Halt}}$ line provides an input to the MPU to allow control of program execution by an outside source. If $\overline{\text{Halt}}$ is high, the MPU will execute the instructions; if it is low, the MPU will go to a halted or idle mode. A response signal, Bus Available (BA) provides an indication of the current MPU status. When BA is low, the MPU is in the process of executing the control program; if BA is high, the MPU has halted and all internal activity has stopped.

When BA is high, the Address Bus, Data Bus, and $\overline{\text{R/W}}$ line will be in a high impedance state, effectively removing the MPU from the system bus. VMA is forced low so that the floating system bus will not activate any device on the bus that is enabled by VMA.



FIGURE 10 – MPU FLOW CHART



- NOTES: 1. Reset is recognized at any position in the flowchart.
 2. Instructions which affect the I-Bit act upon a one-bit buffer register, "ITEMP". This has the effect of delaying any CLEARING of the I-Bit one clock time. Setting the I-Bit, however is not delayed.



While the MPU is halted, all program activity is stopped, and if either an $\overline{\text{NMI}}$ or $\overline{\text{IRQ}}$ interrupt occurs, it will be latched into the MPU and acted on as soon as the MPU is taken out of the halted mode. If a $\overline{\text{Reset}}$ command occurs while the MPU is halted, the following states occur: $\text{VMA} = \text{low}$, $\text{BA} = \text{low}$, Data Bus = high impedance, $\text{R}/\overline{\text{W}} = \text{high}$ (read state), and the Address Bus will contain address FFFF as long as $\overline{\text{Reset}}$ is low. As soon as the $\overline{\text{Halt}}$ line goes high, the MPU will go to locations FFFF and FFFF for the address of the reset routine.

Figure 13 shows the timing relationships involved when halting the MPU. The instruction illustrated is a one byte, 2 cycle instruction such as CLRA . When $\overline{\text{Halt}}$ goes low, the MPU will halt after completing execution of the current instruction. The transition of $\overline{\text{Halt}}$ must occur t_{PCS} before the trailing edge of $\phi 1$ of the last cycle of an instruction (point A of Figure 13). $\overline{\text{Halt}}$ must not go low any time later than the minimum t_{PCS} specified.

The fetch of the OP code by the MPU is the first cycle of the instruction. If $\overline{\text{Halt}}$ had not been low at Point A but went low during $\phi 2$ of that cycle, the MPU would have halted after completion of the following instruction. BA will go high by time t_{BA} (bus available delay time) after the last instruction cycle. At this point in time, VMA is low and $\text{R}/\overline{\text{W}}$, Address Bus, and the Data Bus are in the high impedance state.

To debug programs it is advantageous to step through programs instruction by instruction. To do this, $\overline{\text{Halt}}$ must be brought high for one MPU cycle and then returned low as shown at point B of Figure 13. Again, the transitions of $\overline{\text{Halt}}$ must occur t_{PCS} before the trailing edge of $\phi 1$. BA will go low at t_{BA} after the leading edge of the next $\phi 1$, indicating that the Address Bus, Data Bus, VMA and $\text{R}/\overline{\text{W}}$ lines are back on the bus. A single byte, 2 cycle instruction such as LSR is used for this example also. During the first cycle, the instruction Y is fetched from address $\text{M} + 1$. BA returns high at t_{BA} on the last cycle of the instruction indicating the MPU is off the bus. If instruction Y had been three cycles, the width of the BA low time would have been increased by one cycle.

FIGURE 11 — WAIT INSTRUCTION TIMING

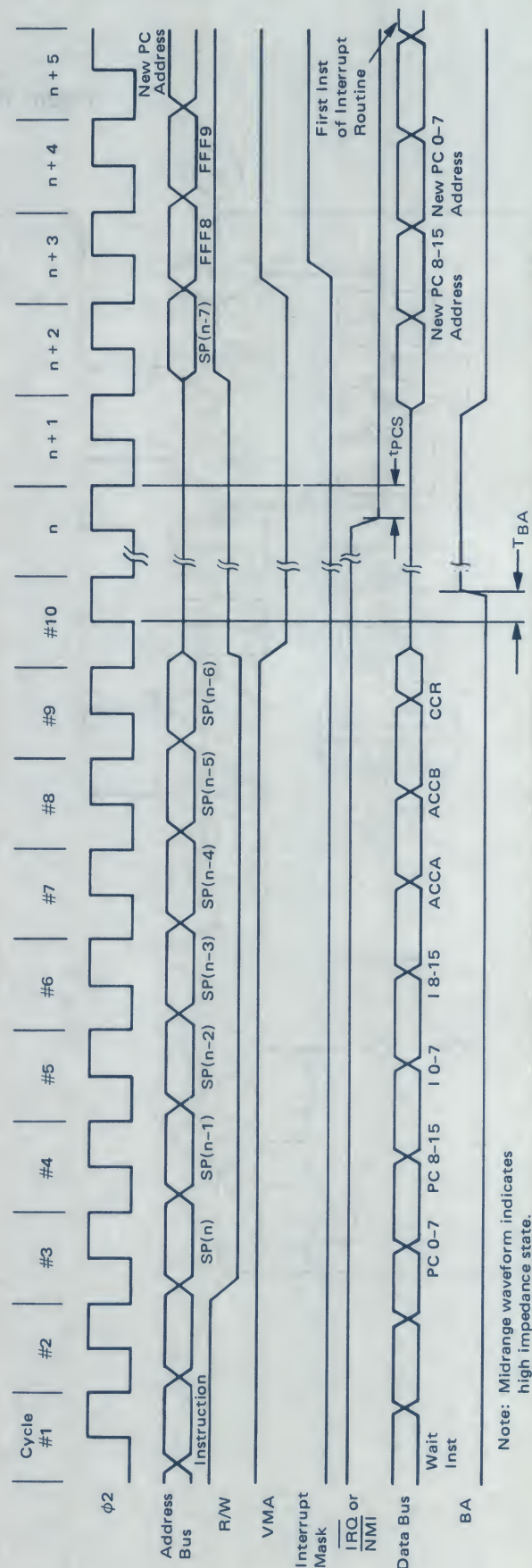
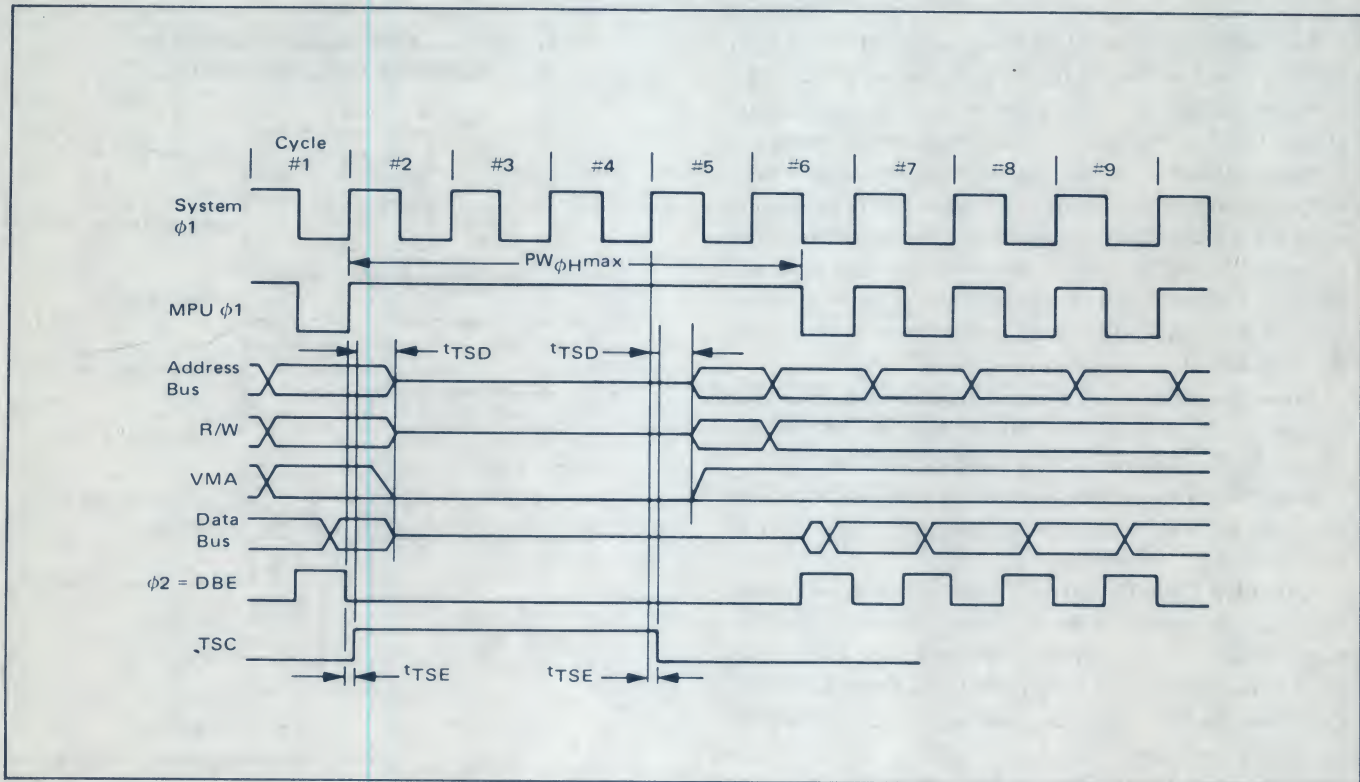
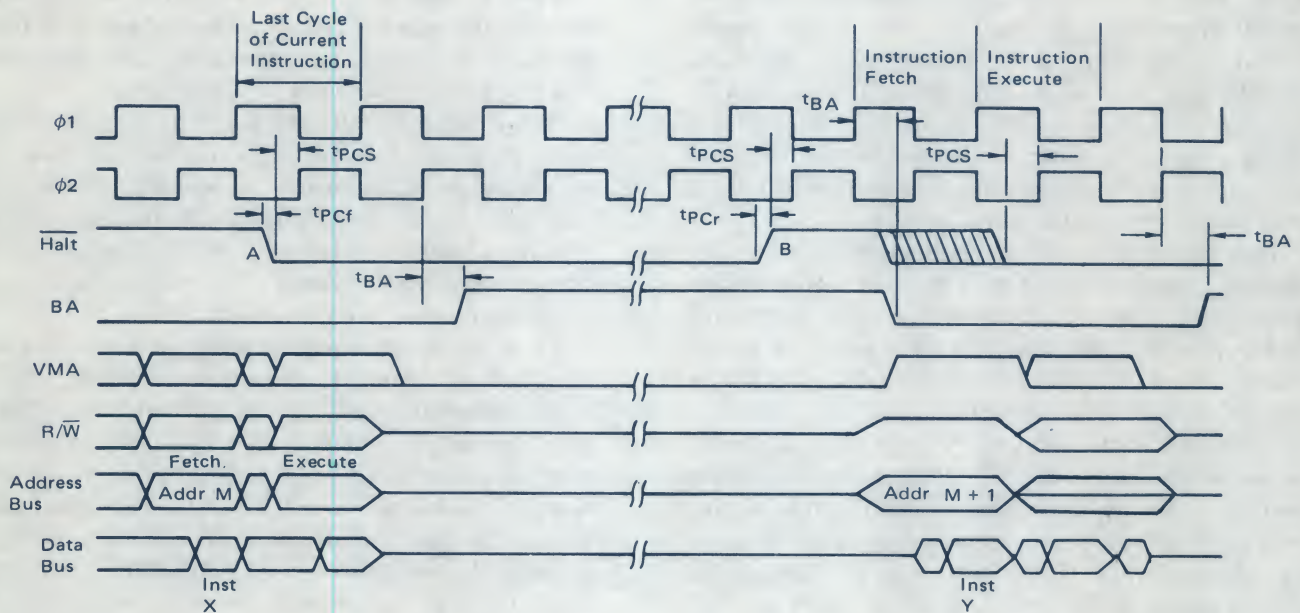


FIGURE 12 – THREE STATE CONTROL TIMING

FIGURE 13 – $\overline{\text{HALT}}$ AND SINGLE INSTRUCTION EXECUTION FOR SYSTEM DEBUG

Note: Midrange waveform indicates high impedance state.



MPU REGISTERS

The MPU has three 16-bit registers and three 8-bit registers available for use by the programmer (Figure 14).

Program Counter — The program counter is a two byte (16 bits) register that points to the current program address.

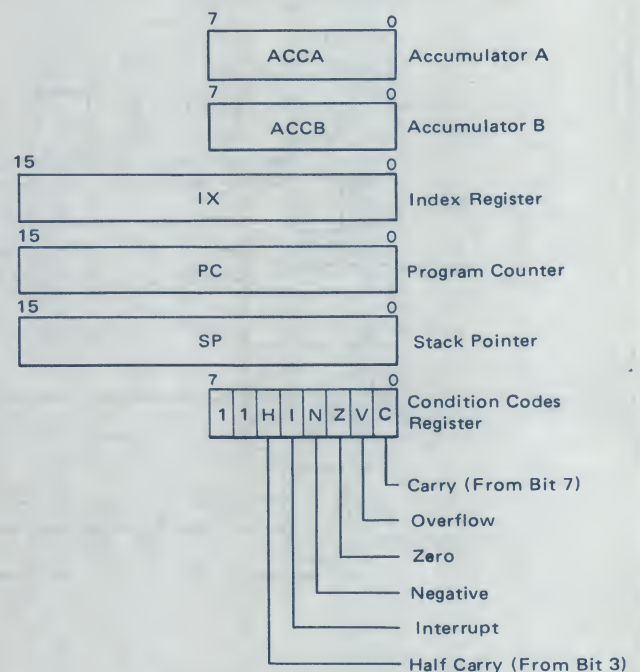
Stack Pointer — The stack pointer is a two byte register that contains the address of the next available location in an external push-down/pop-up stack. This stack is normally a random access Read/Write memory that may have any location (address) that is convenient. In those applications that require storage of information in the stack when power is lost, the stack must be nonvolatile.

Index Register — The index register is a two byte register that is used to store data or a sixteen bit memory address for the Indexed mode of memory addressing.

Accumulators — The MPU contains two 8-bit accumulators that are used to hold operands and results from an arithmetic logic unit (ALU).

Condition Code Register — The condition code register indicates the results of an Arithmetic Logic Unit operation: Negative (N), Zero (Z), Overflow (V), Carry from bit 7 (C), and half carry from bit 3 (H). These bits of the Condition Code Register are used as testable conditions for the conditional branch instructions. Bit 4 is the interrupt mask bit (I). The unused bits of the Condition Code Register (b6 and b7) are ones.

FIGURE 14 — PROGRAMMING MODEL OF THE MICROPROCESSING UNIT



MPU INSTRUCTION SET

The MC6800 instructions are described in detail in the M6800 Programming Manual. This Section will provide a brief introduction and discuss their use in developing MC6800 control programs. The MC6800 has a set of 72 different executable source instructions. Included are binary and decimal arithmetic, logical, shift, rotate, load, store, conditional or unconditional branch, interrupt and stack manipulation instructions.

Each of the 72 executable instructions of the source language assembles into 1 to 3 bytes of machine code. The number of bytes depends on the particular instruction and on the addressing mode. (The addressing modes which are available for use with the various executive instructions are discussed later.)

The coding of the first (or only) byte corresponding to an executable instruction is sufficient to identify the instruction and the addressing mode. The hexadecimal equivalents of the binary codes, which result from the translation of the 72 instructions in all valid modes of addressing, are shown in Table 6. There are 197 valid machine codes, 59 of the 256 possible codes being unassigned.

When an instruction translates into two or three bytes of code, the second byte, or the second and third bytes contain(s) an operand, an address, or information from which an address is obtained during execution.

Microprocessor instructions are often divided into three general classifications: (1) memory reference, so called because they operate on specific memory locations; (2) operating instructions that function without needing a memory reference; (3) I/O instructions for transferring data between the microprocessor and peripheral devices.

In many instances, the MC6800 performs the same operation on both its internal accumulators and the external memory locations. In addition, the MC6800 interfaces adapters (PIA and ACIA) allow the MPU to treat peripheral devices exactly like other memory locations, hence, no I/O instructions as such are required. Because of these features, other classifications are more suitable for introducing the MC6800's instruction set: (1) Accumulator and memory operations; (2) Program control operations; (3) Condition Code Register operations.



TABLE 6 — HEXADECIMAL VALUES OF MACHINE CODES

00	*			40	NEG	A		80	SUB	A	IMM	C0	SUB	B	IMM
01	NOP			41	*			81	CMP	A	IMM	C1	CMP	B	IMM
02	*			42	*			82	SBC	A	IMM	C2	SBC	B	IMM
03	*			43	COM	A		83	*			C3	*		
04	*			44	LSR	A		84	AND	A	IMM	C4	AND	B	IMM
05	*			45	*			85	BIT	A	IMM	C5	BIT	B	IMM
06	TAP			46	ROR	A		86	LDA	A	IMM	C6	LDA	B	IMM
07	TPA			47	ASR	A		87	*			C7	*		
08	INX			48	ASL	A		88	EOR	A	IMM	C8	EOR	B	IMM
09	DEX			49	ROL	A		89	ADC	A	IMM	C9	ADC	B	IMM
0A	CLV			4A	DEC	A		8A	ORA	A	IMM	CA	ORA	B	IMM
0B	SEV			4B	*			8B	ADD	A	IMM	CB	ADD	B	IMM
0C	CLC			4C	INC	A		8C	CPX	A	IMM	CC	*		
0D	SEC			4D	TST	A		8D	BSR		REL	CD	*		
0E	CLI			4E	*			8E	LDS		IMM	CE	LDX		IMM
0F	SEI			4F	CLR	A		8F	*			CF	*		
10	SBA			50	NEG	B		90	SUB	A	DIR	D0	SUB	B	DIR
11	CBA			51	*			91	CMP	A	DIR	D1	CMP	B	DIR
12	*			52	*			92	SBC	A	DIR	D2	SBC	B	DIR
13	*			53	COM	B		93	*			D3	*		
14	*			54	LSR	B		94	AND	A	DIR	D4	AND	B	DIR
15	*			55	*			95	BIT	A	DIR	D5	BIT	B	DIR
16	TAB			56	ROR	B		96	LDA	A	DIR	D6	LDA	B	DIR
17	TBA			57	ASR	B		97	STA	A	DIR	D7	STA	B	DIR
18	*			58	ASL	B		98	EOR	A	DIR	D8	EOR	B	DIR
19	DAA			59	ROL	B		99	ADC	A	DIR	D9	ADC	B	DIR
1A	*			5A	DEC	B		9A	ORA	A	DIR	DA	ORA	B	DIR
1B	ABA			5B	*			9B	ADD	A	DIR	DB	ADD	B	DIR
1C	*			5C	INC	B		9C	CPX		DIR	DC	*		
1D	*			5D	TST	B		9D	*			DD	*		
1E	*			5E	*			9E	LDS		DIR	DE	LDX		DIR
1F	*			5F	CLR	B		9F	STS		DIR	DF	STX		DIR
20	BRA	REL		60	NEG		IND	A0	SUB	A	IND	E0	SUB	B	IND
21	*			61	*			A1	CMP	A	IND	E1	CMP	B	IND
22	BHI	REL		62	*			A2	SBC	A	IND	E2	SBC	B	IND
23	BLS	REL		63	COM		IND	A3	*			E3	*		
24	BCC	REL		64	LSR		IND	A4	AND	A	IND	E4	AND	B	IND
25	BCS	REL		65	*			A5	BIT	A	IND	E5	BIT	B	IND
26	BNE	REL		66	ROR		IND	A6	LDA	A	IND	E6	LDA	B	IND
27	BEQ	REL		67	ASR		IND	A7	STA	A	IND	E7	STA	B	IND
28	BVC	REL		68	ASL		IND	A8	EOR	A	IND	E8	EOR	B	IND
29	BVS	REL		69	ROL		IND	A9	ADC	A	IND	E9	ADC	B	IND
2A	BPL	REL		6A	DEC		IND	AA	ORA	A	IND	EA	ORA	B	IND
2B	BMI	REL		6B	*			AB	ADD	A	IND	EB	ADD	B	IND
2C	BGE	REL		6C	INC		IND	AC	CPX		IND	EC	*		
2D	BLT	REL		6D	TST		IND	AD	JSR		IND	ED	*		
2E	BGT	REL		6E	JMP		IND	AE	LDS		IND	EE	LDX		IND
2F	BLE	REL		6F	CLR		IND	AF	STS		IND	EF	STX		IND
30	TSX			70	NEG		EXT	B0	SUB	A	EXT	F0	SUB	B	EXT
31	INS			71	*			B1	CMP	A	EXT	F1	CMP	B	EXT
32	PUL	A		72	*			B2	SBC	A	EXT	F2	SBC	B	EXT
33	PUL	B		73	COM		EXT	B3	*			F3	*		
34	DES			74	LSR		EXT	B4	AND	A	EXT	F4	AND	B	EXT
35	TXS			75	*			B5	BIT	A	EXT	F5	BIT	B	EXT
36	PSH	A		76	ROR		EXT	B6	LDA	A	EXT	F6	LDA	B	EXT
37	PSH	B		77	ASR		EXT	B7	STA	A	EXT	F7	STA	B	EXT
38	*			78	ASL		EXT	B8	EOR	A	EXT	F8	ADC	B	EXT
39	RTS			79	ROL		EXT	B9	ADC	A	EXT	F9	ADC	B	EXT
3A	*			7A	DEC		EXT	BA	ORA	A	EXT	FA	ORA	B	EXT
3B	RTI			7B	*			BB	ADD	A	EXT	FB	ADD	B	EXT
3C	*			7C	INC		EXT	BC	CPX		EXT	FC	*		
3D	*			7D	TST		EXT	BD	JSR		EXT	FD	*		
3E	WAI			7E	JMP		EXT	BE	LDS		EXT	FE	LDX		EXT
3F	SWI			7F	CLR		EXT	BF	STS		EXT	FF	STX		EXT

- Notes: 1. Addressing Modes: A = Accumulator A IMM = Immediate
 B = Accumulator B DIR = Direct
 REL = Relative
 IND = Indexed
2. Unassigned code indicated by "****"



TABLE 7 — ACCUMULATOR AND MEMORY OPERATIONS

The accumulator and memory operations and their effect on the CCR are shown in Table 7.
Included are Arithmetic Logic, Data Transfer and Data Handling instructions.

		ADDRESSING MODES										BOOLEAN/ARITHMETIC OPERATION										COND. CODE REG.											
		IMMED		DIRECT		INDEX		EXTND		IMPLIED		(All register labels refer to contents)										5		4		3		2		1		0	
OPERATIONS	MNEMONIC	DP	~ =	DP	~ =	OP	~ =	DP	~ =	DP	~ =	DP	~ =											H	I	N	Z	V	C				
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3	A + M → A										↑	↑	↑	↑	↑	↑				
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3	B + M → B										↑	↑	↑	↑	↑	↑				
Add Acmltrs	ABA												1B	2	1	A + B → A										↑	↑	↑	↑	↑	↑		
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3	A + M + C → A										↑	↑	↑	↑	↑	↑				
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3	B + M + C → B										↑	↑	↑	↑	↑	↑				
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3	A · M → A										●	●	●	●	●	●				
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3	B · M → B										●	●	●	●	●	●				
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3	A · M										●	●	●	●	●	●				
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3	B · M										●	●	●	●	●	●				
Clear	CLR							6F	7	2	7F	6	3	00 → M										●	●	●	●	●	●				
	CLRA												4F	2	1	00 → A										●	●	●	●	●	●		
	CLRB												5F	2	1	00 → B										●	●	●	●	●	●		
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3	A - M										●	●	●	●	●	●				
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3	B - M										●	●	●	●	●	●				
Compare Acmltrs	CBA												11	2	1	A - B										●	●	●	●	●	●		
Complement, 1's	COM							63	7	2	73	6	3	M → M										●	●	●	●	●	●				
	COMA												43	2	1	A → A										●	●	●	●	●	●		
	COMB												53	2	1	B → B										●	●	●	●	●	●		
Complement, 2's (Negate)	NEG							60	7	2	70	6	3	00 - M → M										●	●	●	●	●	●				
	NEGA												40	2	1	00 - A → A										●	●	●	●	●	●		
	NEGB												50	2	1	00 - B → B										●	●	●	●	●	●		
Decimal Adjust, A	OAA												19	2	1	Converts Binary Add. of BCD Characters into BCD Format										●	●	●	●	●	●		
Decrement	DEC							6A	7	2	7A	6	3	M - 1 → M										●	●	●	●	●	●				
	OECA												4A	2	1	A - 1 → A										●	●	●	●	●	●		
	OEGB												5A	2	1	B - 1 → B										●	●	●	●	●	●		
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3	A ⊕ M → A										●	●	●	●	●	●				
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3	B ⊕ M → B										●	●	●	●	●	●				
Increment	INC							6C	7	2	7C	6	3	M + 1 → M										●	●	●	●	●	●				
	INCA												4C	2	1	A + 1 → A										●	●	●	●	●	●		
	INCB												5C	2	1	B + 1 → B										●	●	●	●	●	●		
Load Acmltr	LDA	86	2	2	96	3	2	A6	5	2	B6	4	3	M → A										●	●	●	●	●	●				
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3	M → B										●	●	●	●	●	●				
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3	A + M → A										●	●	●	●	●	●				
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3	B + M → B										●	●	●	●	●	●				
Push Data	PSHA												36	4	1	A → Msp, SP - 1 → SP										●	●	●	●	●	●		
	PSHB												37	4	1	B → Msp, SP - 1 → SP										●	●	●	●	●	●		
Pull Data	PULA												32	4	1	SP + 1 → SP, Msp → A										●	●	●	●	●	●		
	PULB												33	4	1	SP + 1 → SP, Msp → B										●	●	●	●	●	●		
Rotate Left	ROL							69	7	2	79	6	3	M										●	●	●	●	●	●				
	ROLA												49	2	1	A										●	●	●	●	●	●		
	ROLB												59	2	1	B										●	●	●	●	●	●		
Rotate Right	ROR							66	7	2	76	6	3	M										●	●	●	●	●	●				
	RORA												46	2	1	A										●	●	●	●	●	●		
	RORB												56	2	1	B										●	●	●	●	●	●		
Shift Left, Arithmetic	ASL							68	7	2	78	6	3	M										●	●	●	●	●	●				
	ASLA												48	2	1	A										●	●	●	●	●	●		
	ASLB												58	2	1	B										●	●	●	●	●	●		
Shift Right, Arithmetic	ASR							67	7	2	77	6	3	M										●	●	●	●	●	●				
	ASRA												47	2	1	A										●	●	●	●	●	●		
	ASRB												57	2	1	B										●	●	●	●	●	●		
Shift Right, Logic	LSR							64	7	2	74	6	3	M										●	●	●	●	●	●				
	LSRA												44	2	1	A										●	●	●	●	●	●		
	LSRB												54	2	1	B										●	●	●	●	●	●		
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3	A → M										●	●	●	●	●	●				
	STAB				D7	4	2	E7	6	2	F7	5	3	B → M										●	●	●	●	●	●				
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3	A - M → A										●	●	●	●	●	●				
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3	B - M → B										●	●	●	●	●	●				
Subtract Acmltrs.	SBA												10	2	1	A - B → A										●	●	●	●	●	●		
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3	A - M - C → A										●	●	●	●	●	●				
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3	B - M - C → B										●	●	●	●	●	●				
Transfer Acmltrs	TAB												16	2	1	A → B										●	●	●	●	●	●		
	TBA												17	2	1	B → A										●	●	●	●	●	●		
Test, Zero or Minus	TST							6D	7	2	7D	6	3	M - 00										●	●	●	●	●	●				
	TSTA												4D	2	1	A - 00										●	●	●	●	●	●		
	TSTB												5D	2	1	B - 00										●	●	●	●	●	●		

LEGEND:

OP Operation Code (Hexadecimal);
~ Number of MPU Cycles;
Number of Program Bytes;
+ Arithmetic Plus;
- Arithmetic Minus;
• Boolean AND;
Msp Contents of memory location pointed to by Stack Pointer;

+ Boolean Inclusive OR;
⊕ Boolean Exclusive OR;
M Complement of M;
→ Transfer Into;
0 Bit = Zero;
00 Byte = Zero;

CONDITION CODE SYMBOLS:

H Half-carry from bit 3;
I Interrupt mask;
N Negative (sign bit);
Z Zero (byte);
V Overflow, 2's complement;
C Carry from bit 7;
R Reset Always;
S Set Always;
† Test and set if true, cleared otherwise;
• Not Affected

Note — Accumulator addressing mode instructions are included in the column for IMPLIED addressing



MOTOROLA Semiconductor Products Inc.

TABLE 7 – CONTINUED

CONDITION CODE REGISTER NOTES: (Bit set if test is true and cleared otherwise)

1	(Bit V)	Test: Result = 10000000?
2	(Bit C)	Test: Result = 00000000?
3	(Bit C)	Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4	(Bit V)	Test: Operand = 10000000 prior to execution?
5	(Bit V)	Test: Operand = 01111111 prior to execution?
6	(Bit V)	Test: Set equal to result of NQC after shift has occurred.

PROGRAM CONTROL OPERATIONS

Program Control operation can be subdivided into two categories: (1) Index Register/Stack Pointer instructions; (2) Jump and Branch operations.

Index Register/Stack Pointer Operations

The instructions for direct operation on the MPU's Index Register and Stack Pointer are summarized in Table 8. Decrement (DEX, DES), increment (INX, INS), load (LDX, LDS), and store (STX, STS) instructions are provided for both. The Compare instruction, CPX, can be used to compare the Index Register to a 16-bit value and update the Condition Code Register accordingly.

The TSX instruction causes the Index Register to be loaded with the address of the last data byte put onto the "stack". The TXS instruction loads the Stack Pointer with a value equal to one less than the current contents

of the Index Register. This causes the next byte to be pulled from the "stack" to come from the location indicated by the Index Register. The utility of these two instructions can be clarified by describing the "stack" concept relative to the M6800 system.

The "stack" can be thought of as a sequential list of data stored in the MPU's read/write memory. The Stack Pointer contains a 16-bit memory address that is used to access the list from one end on a last-in-first-out (LIFO) basis in contrast to the random access mode used by the MPU's other addressing modes.

The M6800 instruction set and interrupt structure allow extensive use of the stack concept for efficient handling of data movement, subroutines and interrupts. The instructions can be used to establish one or more "stacks" anywhere in read/write memory. Stack length is limited only by the amount of memory that is made available.

TABLE 8 – INDEX REGISTER AND STACK POINTER INSTRUCTIONS

																	COND. CODE REG.						
		IMMED			DIRECT			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
POINTER OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BOOLEAN/ARITHMETIC OPERATION	H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M + 1)$	•	•	①	↑	②	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↑	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↑	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M + 1) \rightarrow X_L$	•	•	•	③	↑	R
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$	•	•	•	③	↑	R
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M + 1)$	•	•	•	③	↑	R
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	•	③	↑	R
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

- ① (Bit N) Test: Sign bit of most significant (MS) byte of result = 1?
- ② (Bit V) Test: 2's complement overflow from subtraction of ms bytes?
- ③ (Bit N) Test: Result less than zero? (Bit 15 = 1)



FIGURE 15 – STACK OPERATION, PUSH INSTRUCTION

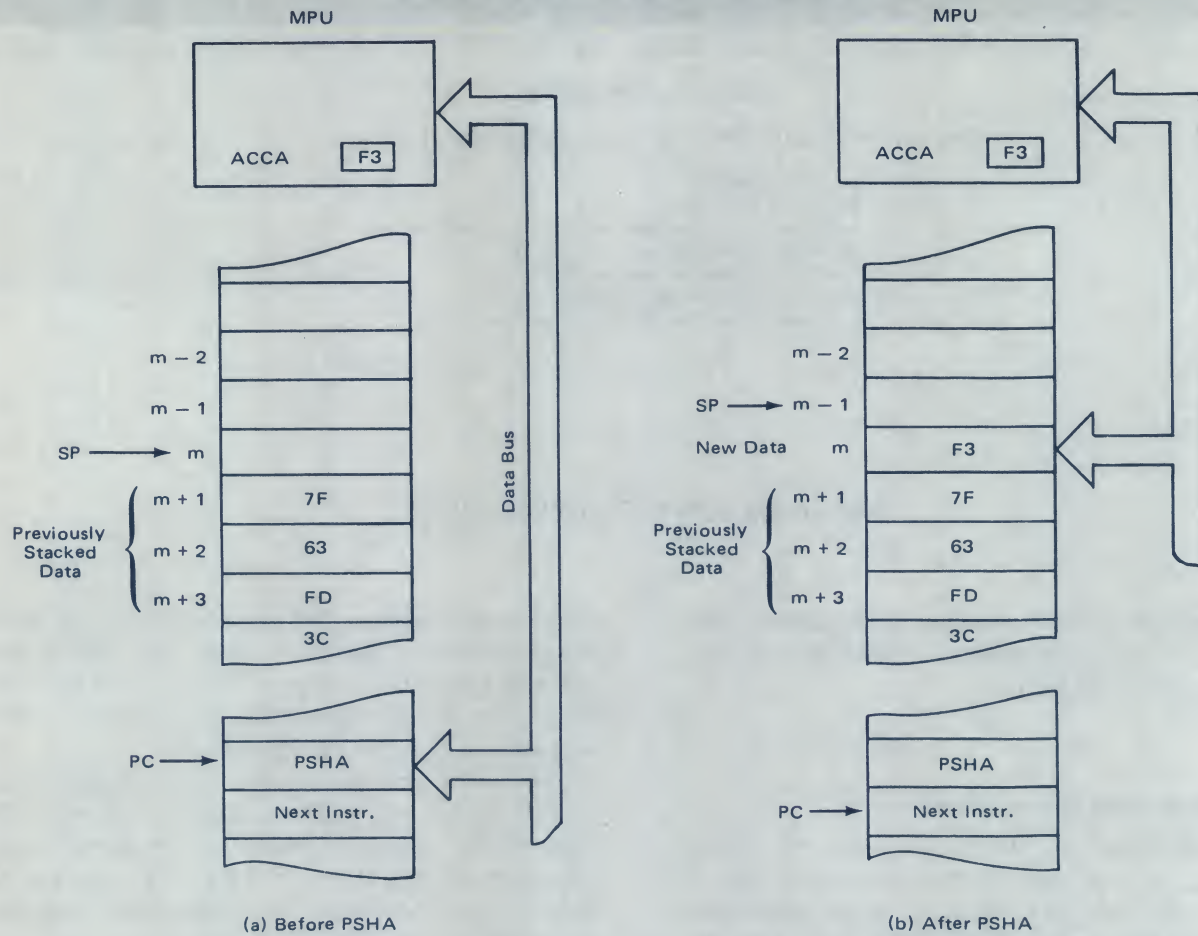
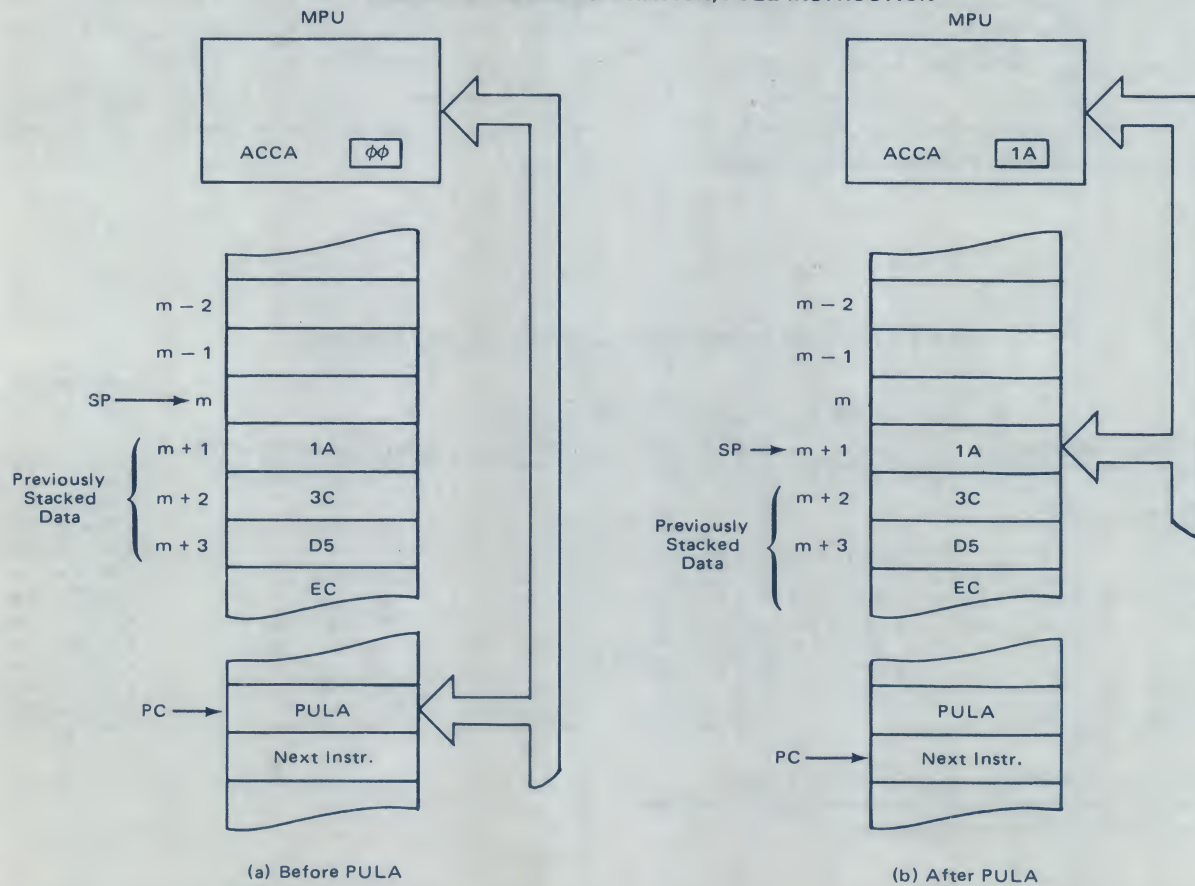


FIGURE 16 – STACK OPERATION, PULL INSTRUCTION



Execution of the Jump Instruction, JMP, and Branch Always, BRA, affects program flow as shown in Figure 17. When the MPU encounters the Jump (Indexed) instruction, it adds the offset to the value in the Index Register and uses the result as the address of the next instruction to be executed. In the extended addressing mode, the address of the next instruction to be executed is fetched from the two locations immediately following the JMP instruction. The Branch Always (BRA) instruction is similar to the JMP (extended) instruction except that the relative addressing mode applies and the branch is limited to the range within -125 or $+127$ bytes of the branch instruction itself. The opcode for the BRA instruction requires one less byte than JMP (extended) but takes one more cycle to execute.

The effect on program flow for the Jump to Subroutine (JSR) and Branch to Subroutine (BSR) is shown in Figures 18 through 20. Note that the Program Counter is properly incremented to be pointing at the correct return address before it is stacked. Operation of the Branch to Subroutine and Jump to Subroutine (extended) instruction is similar except for the range. The BSR instruction requires less opcode than JSR (2 bytes versus 3 bytes)

and also executes one cycle faster than JSR. The Return from Subroutine, RTS, is used at the end of a subroutine to return to the main program as indicated in Figure 21.

The effect of executing the Software Interrupt, SWI, and the Wait for Interrupt, WAI, and their relationship to the hardware interrupts is shown in Figure 22. SWI causes the MPU contents to be stacked and then fetches the starting address of the interrupt routine from the memory locations that respond to the addresses FFFA and FFFB. Note that as in the case of the subroutine instructions, the Program Counter is incremented to point at the correct return address before being stacked. The Return from Interrupt instruction, RTI, (Figure 22) is used at the end of an interrupt routine to restore control to the main program. The SWI instruction is useful for inserting break points in the control program, that is, it can be used to stop operation and put the MPU registers in memory where they can be examined. The WAI instruction is used to decrease the time required to service a hardware interrupt; it stacks the MPU contents and then waits for the interrupt to occur, effectively removing the stacking time from a hardware interrupt sequence.

FIGURE 17 – PROGRAM FLOW FOR JUMP AND BRANCH INSTRUCTIONS

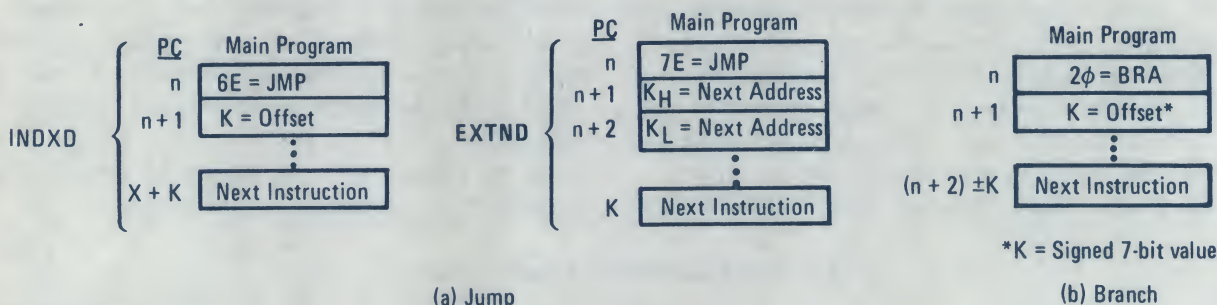


FIGURE 18 – PROGRAM FLOW FOR BSR

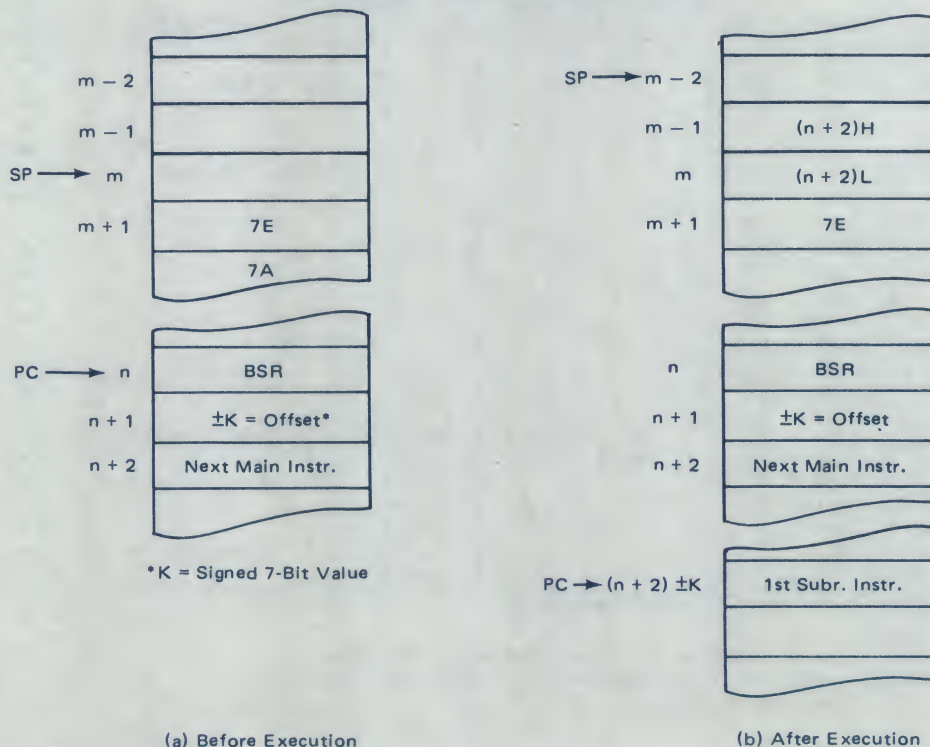


FIGURE 19 – PROGRAM FLOW FOR JSR (EXTENDED)

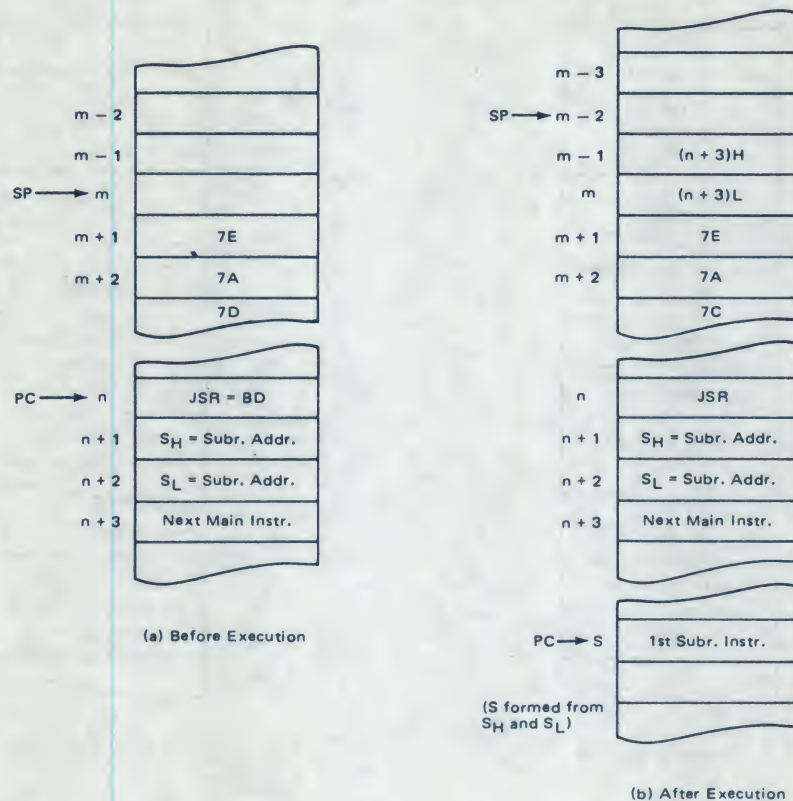


FIGURE 20 – PROGRAM FLOW FOR JSR (INDEXED)

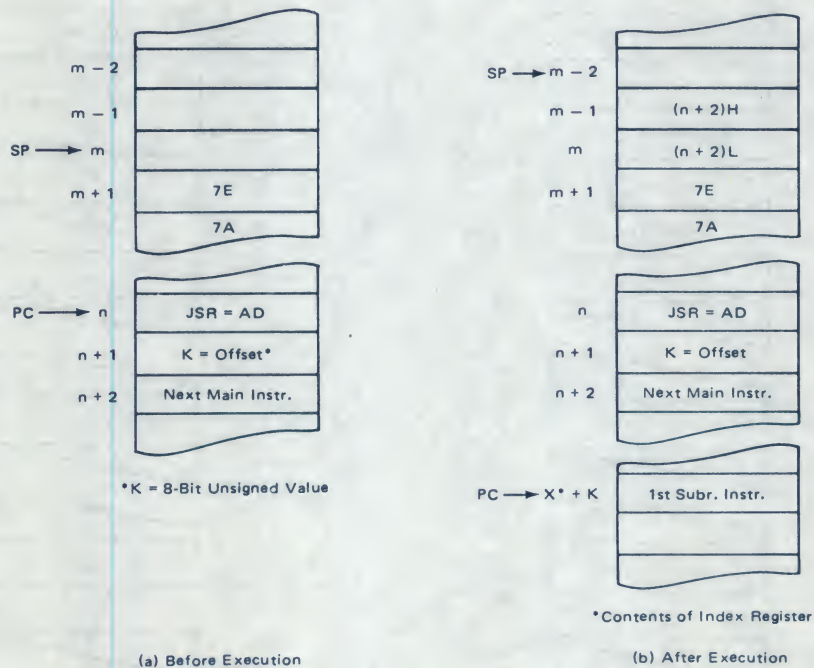


FIGURE 21 – PROGRAM FLOW FOR RTS

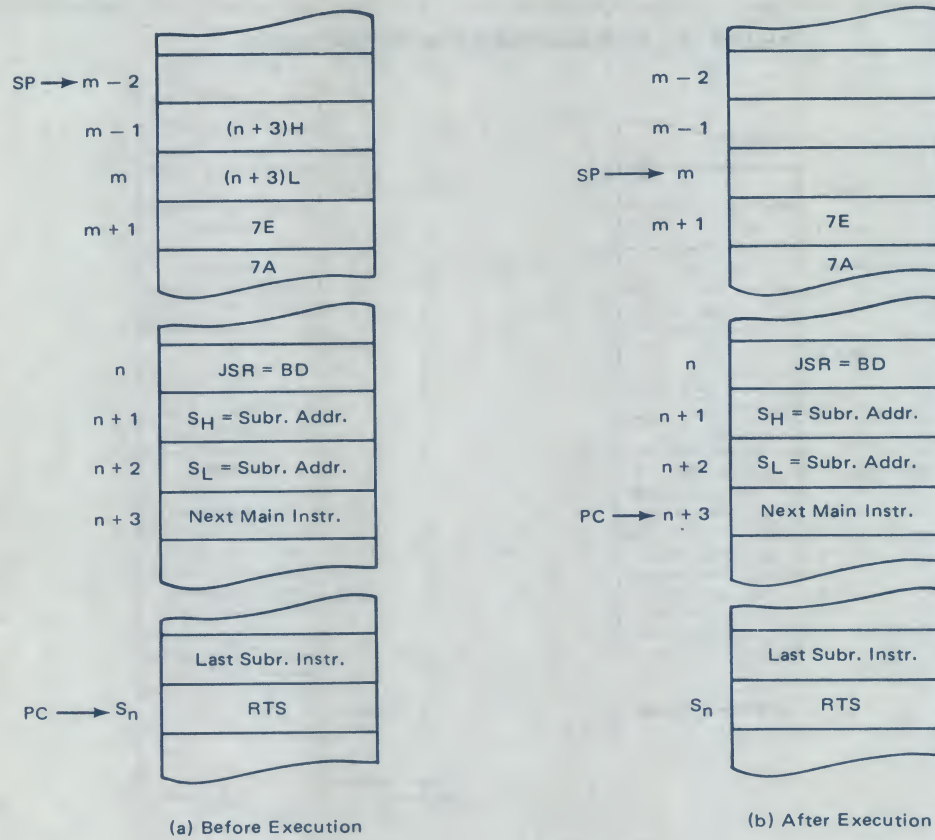


FIGURE 22 – PROGRAM FLOW FOR RTI

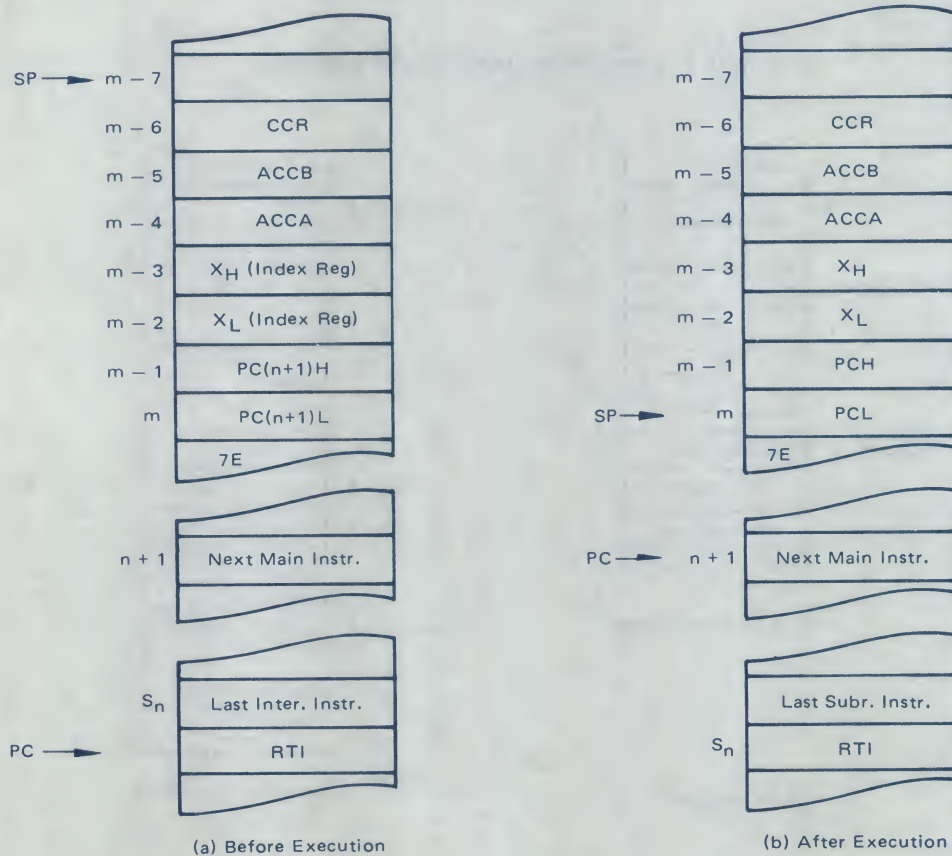


FIGURE 23 – PROGRAM FLOW FOR INTERRUPTS

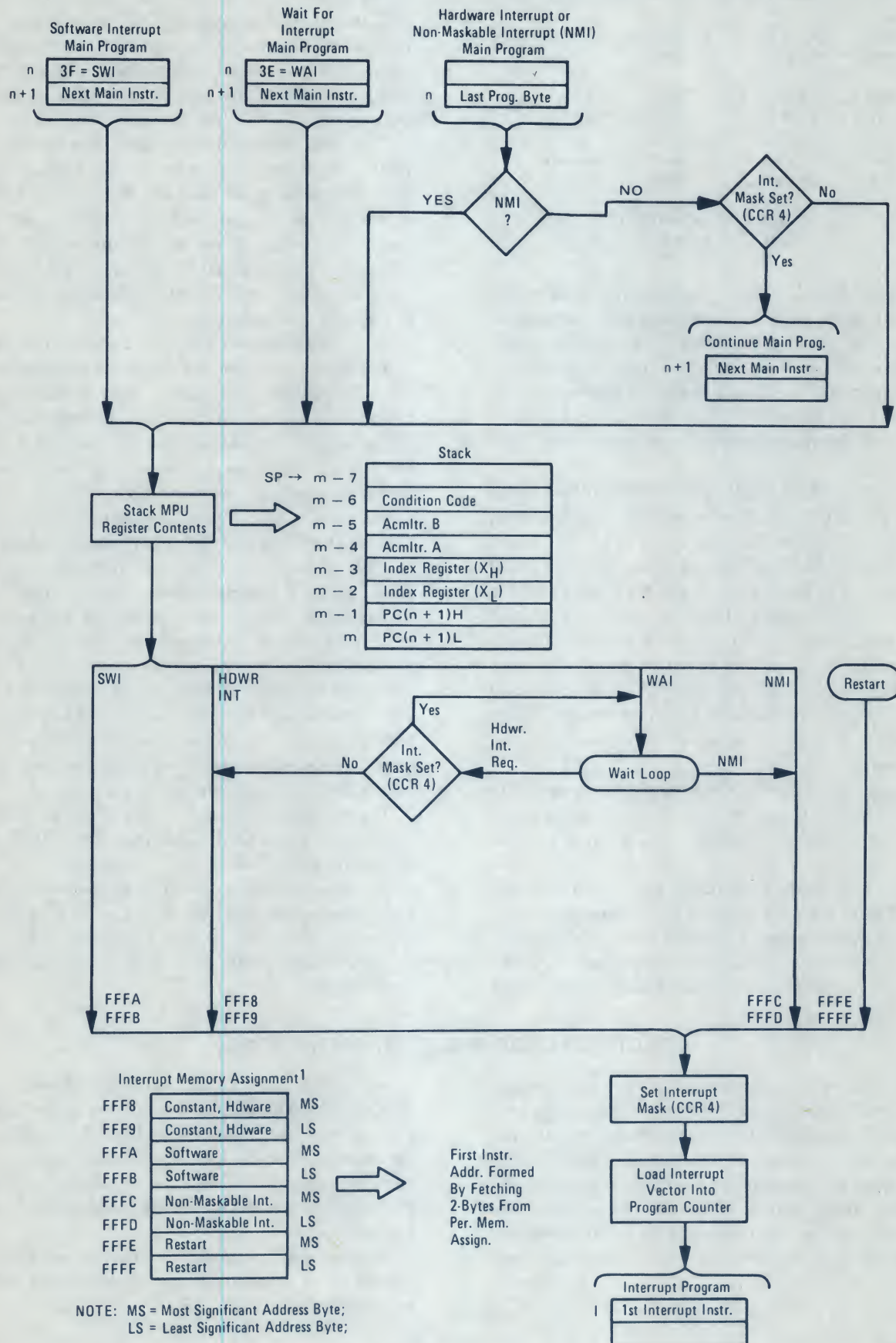


FIGURE 24 — CONDITIONAL BRANCH INSTRUCTIONS

BMI :	N = 1 ;	BEQ :	Z = 1 ;
BPL :	N = ϕ ;	BNE :	Z = ϕ ;
BVC :	V = ϕ ;	BCC :	C = ϕ ;
BVS :	V = 1 ;	BCS :	C = 1 ;
BHI :	C + Z = ϕ ;	BLT :	N \oplus V = 1 ;
BLS :	C + Z = 1 ;	BGE :	N \oplus V = ϕ ;
BLE :	Z + (N \oplus V) = 1 ;		
BGT :	Z + (N \oplus V) = ϕ ;		

The conditional branch instructions, Figure 24, consists of seven pairs of complementary instructions. They are used to test the results of the preceding operation and either continue with the next instruction in sequence (test fails) or cause a branch to another point in the program (test succeeds).

Four of the pairs are used for simple tests of status bits N, Z, V, and C:

1. Branch on Minus (BMI) and Branch On Plus (BPL) tests the sign bit, N, to determine if the previous result was negative or positive, respectively.

2. Branch On Equal (BEQ) and Branch On Not Equal (BNE) are used to test the zero status bit, Z, to determine whether or not the result of the previous operation was equal to zero. These two instructions are useful following a Compare (CMP) instruction to test for equality between an accumulator and the operand. They are also used following the Bit Test (BIT) to determine whether or not the same bit positions are set in an accumulator and the operand.

3. Branch On Overflow Clear (BVC) and Branch On Overflow Set (BVS) tests the state of the V bit to determine if the previous operation caused an arithmetic overflow.

4. Branch On Carry Clear (BCC) and Branch On Carry Set (BCS) tests the state of the C bit to determine if the previous operation caused a carry to occur. BCC and BCS are useful for testing relative magnitude when the values being tested are regarded as unsigned binary numbers, that

is, the values are in the range 00 (lowest) to FF (highest). BCC following a comparison (CMP) will cause a branch if the (unsigned) value in the accumulator is higher than or the same as the value of the operand. Conversely, BCS will cause a branch if the accumulator value is lower than the operand.

The fifth complementary pair, Branch On Higher (BHI) and Branch On Lower or Same (BLS) are in a sense complements to BCC and BCS. BHI tests for both C and Z = 0; if used following a CMP, it will cause a branch if the value in the accumulator is higher than the operand. Conversely, BLS will cause a branch if the unsigned binary value in the accumulator is lower than or the same as the operand.

The remaining two pairs are useful in testing results of operations in which the values are regarded as signed two's complement numbers. This differs from the unsigned binary case in the following sense: In unsigned, the orientation is higher or lower; in signed two's complement, the comparison is between larger or smaller where the range of values is between -128 and +127.

Branch On Less Than Zero (BLT) and Branch On Greater Than Or Equal Zero (BGE) test the status bits for $N \oplus V = 1$ and $N \oplus V = 0$, respectively. BLT will always cause a branch following an operation in which two negative numbers were added. In addition, it will cause a branch following a CMP in which the value in the accumulator was negative and the operand was positive. BLT will never cause a branch following a CMP in which the accumulator value was positive and the operand negative. BGE, the complement to BLT, will cause a branch following operations in which two positive values were added or in which the result was zero.

The last pair, Branch On Less Than Or Equal Zero (BLE) and Branch On Greater Than Zero (BGT) test the status bits for $Z \oplus (N + V) = 1$ and $Z \oplus (N + V) = 0$, respectively. The action of BLE is identical to that for BLT except that a branch will also occur if the result of the previous result was zero. Conversely, BGT is similar to BGE except that no branch will occur following a zero result.

CONDITION CODE REGISTER OPERATIONS

The Condition Code Register (CCR) is a 6-bit register within the MPU that is useful in controlling program flow during system operation. The bits are defined in Figure 25.

The instructions shown in Table 10 are available to the user for direct manipulation of the CCR. In addition, the MPU automatically sets or clears the appropriate status bits as many of the other instructions on the condition code register was indicated as they were introduced.

A CLI-WAI instruction sequence operated properly with early M6800 processors only if the preceding instruction was odd. (Least Significant Bit = 1.) Similarly it was advisable to precede any SEI instruction with an odd opcode—such as NOP. These precautions are not necessary for M6800 processors indicating manufacture in November, 1977 or later.

Systems which require an interrupt window to be opened under program control should use a CLI-NOP-SEI sequence rather than CLI-SEI.



FIGURE 25 – CONDITION CODE REGISTER BIT DEFINITION

b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
H	I	N	Z	V	C

H = Half-carry; set whenever a carry from b₃ to b₄ of the result is generated by ADD, ABA, ADC; cleared if no b₃ to b₄ carry; not affected by other instructions.

I = Interrupt Mask; set by hardware or software interrupt or SEI instruction; cleared by CLI instruction. (Normally not used in arithmetic operations.) Restored to a zero as a result of an RTI instruction if I_m stored on the stack is low.

N = Negative; set if high order bit (b₇) of result is set; cleared otherwise

Z = Zero; set if result = 0; cleared otherwise.

V = Overflow; set if there was arithmetic overflow as a result of the operation; cleared otherwise.

C = Carry; set if there was a carry from the most significant bit (b₇) of the result; cleared otherwise.

TABLE 10 – CONDITION CODE REGISTER INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED			BOOLEAN OPERATION	COND. CODE REG.					
		OP	~	#		5	4	3	2	1	0
						H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Accmltr A → CCR	TAP	06	2	1	A → CCR	①					
CCR → Accmltr A	TPA	07	2	1	CCR → A						

R = Reset

S = Set

• = Not affected

① (ALL) Set according to the contents of Accumulator A.

ADDRESSING MODES

The MPU operates on 8-bit binary numbers presented to it via the Data Bus. A given number (byte) may represent either data or an instruction to be executed, depending on where it is encountered in the control program. The M6800 has 72 unique instructions, however, it recognizes and takes action on 197 of the 256 possibilities that can occur using an 8-bit word length. This larger number of instructions results from the fact that many of the executive instructions have more than one addressing mode.

These addressing modes refer to the manner in which the program causes the MPU to obtain its instructions and data. The programmer must have a method for addressing the MPU's internal registers and all of the external memory locations.

Selection of the desired addressing mode is made by the user as the source statements are written. Translation into appropriate opcode then depends on the method used. If manual translation is used, the addressing mode is inherent in the opcode. For example, the Immediate,



Direct, Indexed, and Extended modes may all be used with the ADD instruction. The proper mode is determined by selecting (hexidecimal notation) 8B, 9B, AB, or BB, respectively.

The source statement format includes adequate information for the selection if an assembler program is used to generate the opcode. For instance, the Immediate mode is selected by the Assembler whenever it encounters the "#" symbol in the operand field. Similarly, an "X" in the operand field causes the Indexed mode to be selected. Only the Relative mode applies to the branch

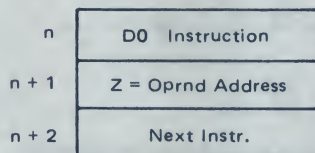
instructions, therefore, the mnemonic instruction itself is enough for the Assembler to determine addressing mode.

For the instructions that use both Direct and Extended modes, the Assembler selects the Direct mode if the operand value is in the range 0-255 and Extended otherwise. There are a number of instructions for which the Extended mode is valid but the Direct is not. For these instructions, the Assembler automatically selects the Extended mode even if the operand is in the 0-255 range. The addressing modes are summarized in Figure 26.

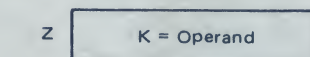
FIGURE 26 — ADDRESSING MODE SUMMARY

Direct:

Example: SUBB Z
Addr. Range = 0-255

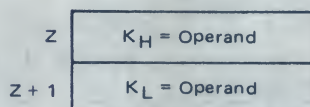


(K = One-Byte Oprnd)



OR

(K = Two-Byte Oprnd)

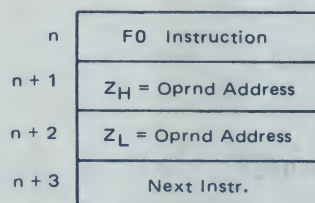


1 If $Z \leq 255$, Assembler Select Direct Mode
If $Z > 255$, Extended Mode is selected

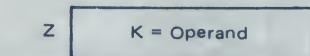
Extended:

Example: CMPA Z

Addr. Range:
1 256-65535

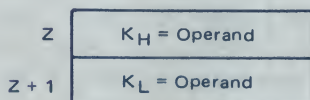


(K = One-Byte Oprnd)

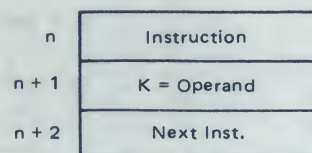


OR

(K = Two-Byte Oprnd)

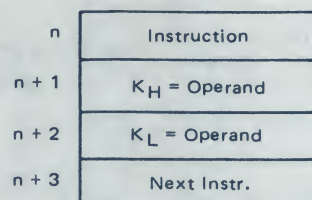
**Immediate:**

Example: LDAA #K
(K = One-Byte Oprnd)



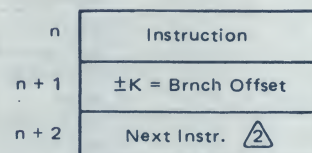
OR

(K = Two-Byte Oprnd)
(CPX, LDX, and LDS)

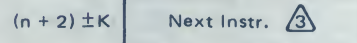
**Relative:**

Example: BNE K

(K = Signed 7-Bit Value)



Addr. Range:
-125 to +129
Relative to n.

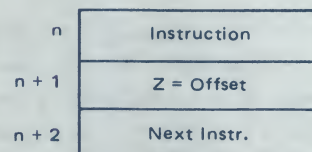


2 If Brnch Tst False, 3 If Brnch Tst True.

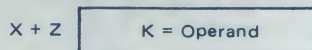
Indexed:

Example: ADDA Z, X

Addr. Range:
0-255 Relative to
Index Register, X



(Z = 8-Bit Unsigned
Value)



Inherent (Includes "Accumulator Addressing" Mode)

The successive fields in a statement are normally separated by one or more spaces. An exception to this rule occurs for instructions that use dual addressing in the operand field and for instructions that must distinguish between the two accumulators. In these cases, A and B are "operands" but the space between them and the operator may be omitted. This is commonly done, resulting in apparent four character mnemonics for those instructions.

The addition instruction, ADD, provides an example of dual addressing in the operand field:

	Operator	Operand	Comment
or	ADDA	MEM12	ADDCONTENTSOF MEM12TO ACCA
	ADDB	MEM12	ADDCONTENTSOF MEM12TO ACB

The example used earlier for the test instruction, TST, also applies to the accumulators and uses the "accumulator addressing mode" to designate which of the two accumulators is being tested:

	Operator	Comment
	TSTB	TEST CONTENTS OF ACCB
or	TSTA	TEST CONTENTS OF ACCA

A number of the instructions either alone or together with an accumulator operand contain all of the address information that is required, that is, "inherent" in the instruction itself. For instance, the instruction ABA causes the MPU to add the contents of accumulators A and B together and place the result in accumulator A. The instruction INCB, another example of "accumulator addressing", causes the contents of accumulator B to be increased by one. Similarly, INX, increment the Index Register, causes the contents of the Index Register to be increased by one.

Program flow for instructions of this type is illustrated in Figures 27 and 28. In these figures, the general case is shown on the left and a specific example is shown on the right. Numerical examples are in decimal notation. Instructions of this type require only one byte of opcode. Cycle-by-cycle operation of the inherent mode is shown in Table 11.

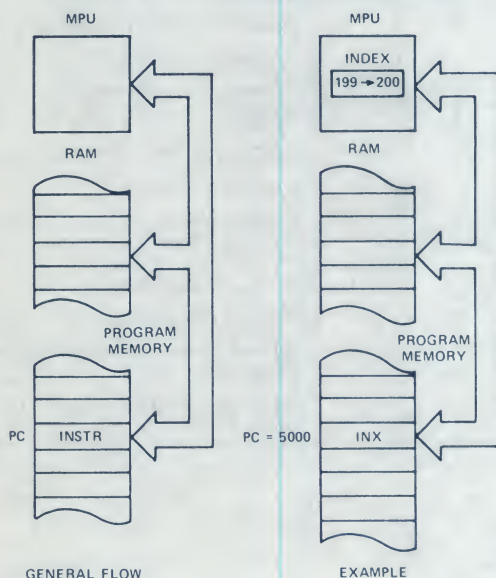
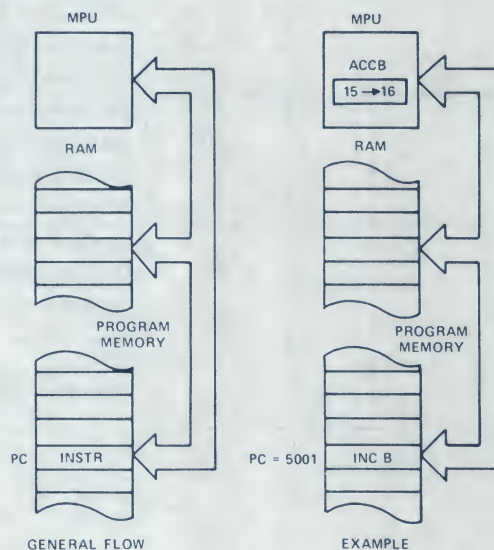
FIGURE 27 – INHERENT ADDRESSING**FIGURE 28 – ACCUMULATOR ADDRESSING**

TABLE 11 – INHERENT MODE CYCLE BY CYCLE OPERATION

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
ABA DAA SEC ASL DEC SEI ASR INC SEV CBA LSR TAB CLC NEG TAP CLI NOP TBA CLR ROL TPA CLV ROR TST COM SBA	2	1 2	1 1	Op Code Address Op Code Address + 1	1 1	Op Code Op Code of Next Instruction
DES DEX INS INX	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Previous Register Contents New Register Contents	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Irrelevant Data (Note 1)
PSH	4	1 2 3 4	1 1 1 0	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer – 1	1 1 0 1	Op Code Op Code of Next Instruction Accumulator Data Accumulator Data
PUL	4	1 2 3 4	1 1 0 1	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer + 1	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Operand Data from Stack
TSX	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Stack Pointer New Index Register	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Irrelevant Data (Note 1)
TXS	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Index Register New Stack Pointer	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data Irrelevant Data
RTS	5	1 2 3 4 5	1 1 0 1 1	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer + 1 Stack Pointer + 2	1 1 1 1 1	Op Code Irrelevant Data (Note 2) Irrelevant Data (Note 1) Address of Next Instruction (High Order Byte) Address of Next Instruction (Low Order Byte)



TABLE 11 — INHERENT MODE CYCLE BY CYCLE OPERATION (Continued)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
WAI	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer - 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer - 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer - 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer - 4	0	Contents of Accumulator A
		8	1	Stack Pointer - 5	0	Contents of Accumulator B
		9	1	Stack Pointer - 6 (Note 3)	1	Contents of Cond. Code Register
RTI	10	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 2)
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Contents of Cond. Code Register from Stack
		5	1	Stack Pointer + 2	1	Contents of Accumulator B from Stack
		6	1	Stack Pointer + 3	1	Contents of Accumulator A from Stack
		7	1	Stack Pointer + 4	1	Index Register from Stack (High Order Byte)
		8	1	Stack Pointer + 5	1	Index Register from Stack (Low Order Byte)
		9	1	Stack Pointer + 6	1	Next Instruction Address from Stack (High Order Byte)
		10	1	Stack Pointer + 7	1	Next Instruction Address from Stack (Low Order Byte)
SWI	12	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 1)
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer - 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer - 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer - 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer - 4	0	Contents of Accumulator A
		8	1	Stack Pointer - 5	0	Contents of Accumulator B
		9	1	Stack Pointer - 6	0	Contents of Cond. Code Register
		10	0	Stack Pointer - 7	1	Irrelevant Data (Note 1)
		11	1	Vector Address FFFA (Hex)	1	Address of Subroutine (High Order Byte)
		12	1	Vector Address FFFB (Hex)	1	Address of Subroutine (Low Order Byte)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

Note 2. Data is ignored by the MPU.

Note 3. While the MPU is waiting for the interrupt, Bus Available will go high indicating the following states of the control lines: VMA is low; Address Bus, R/W, and Data Bus are all in the high impedance state.



Immediate Addressing Mode — In the Immediate addressing mode, the operand is the value that is to be operated on. For instance, the instruction

Operator	Operand	Comment
LDA A	#25	LOAD 25 INTO ACCA

causes the MPU to "immediately load accumulator A with the value 25"; no further address reference is required. The Immediate mode is selected by preceding the operand value with the "#" symbol. Program flow for this addressing mode is illustrated in Figure 29.

The operand format allows either properly defined symbols or numerical values. Except for the instructions CPX, LDX, and LDS, the operand may be any value in the range 0 to 255. Since Compare Index Register (CPX), Load Index Register (LDX), and Load Stack Pointer (LDS), require 16-bit values, the immediate mode for

these three instructions require two-byte operands. In the Immediate addressing mode, the "address" of the operand is effectively the memory location immediately following the instruction itself. Table 12 shows the cycle-by-cycle operation for the immediate addressing mode.

Direct and Extended Addressing Modes — In the Direct and Extended modes of addressing, the operand field of the source statement is the *address* of the value that is to be operated on. The Direct and Extended modes differ only in the range of memory locations to which they can direct the MPU. Direct addressing generates a single 8-bit operand and, hence, can address only memory locations 0 through 255; a two byte operand is generated for Extended addressing, enabling the MPU to reach the remaining memory locations, 256 through 65535. An example of Direct addressing and its effect on program flow is illustrated in Figure 30.

FIGURE 29 — IMMEDIATE ADDRESSING MODE

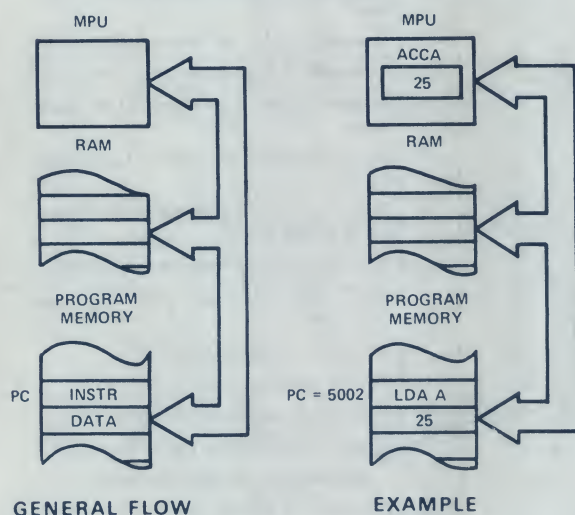


FIGURE 30 — DIRECT ADDRESSING MODE

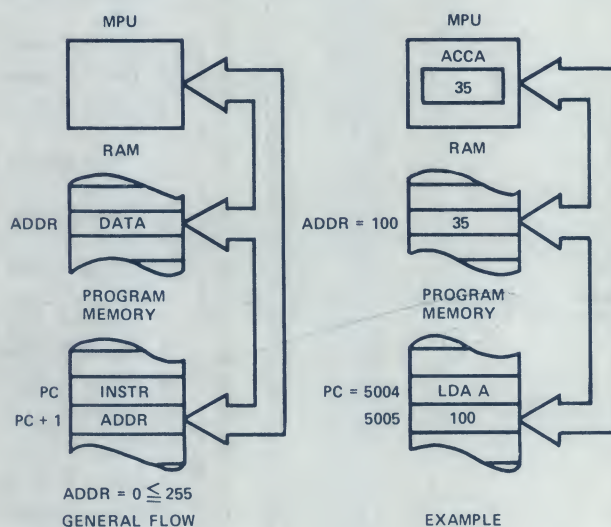


TABLE 12 — IMMEDIATE MODE CYCLE BY CYCLE OPERATION

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data
CPX LDS LDX	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data (High Order Byte)
		3	1	Op Code Address + 2	1	Operand Data (Low Order Byte)



The MPU, after encountering the opcode for the instruction LDAA (Direct) at memory location 5004 (Program Counter = 5004), looks in the next location, 5005, for the address of the operand. It then sets the program counter equal to the value found there (100 in the example) and fetches the operand, in this case a value to be loaded into accumulator A, from that location. For instructions requiring a two-byte operand such as LDX (load the Index Register), the operand bytes would be retrieved from locations 100 and 101. Table 13 shows the cycle-by-cycle operation for the direct mode of addressing.

Extended addressing, Figure 31, is similar except that a two-byte address is obtained from locations 5007 and

5008 after the LDAB (Extended) opcode shows up in location 5006. Extended addressing can be thought of as the "standard" addressing mode, that is, it is a method of reaching anyplace in memory. Direct addressing, since only one address byte is required, provides a faster method of processing data and generates fewer bytes of control code. In most applications, the direct addressing range, memory locations 0-255, are reserved for RAM. They are used for data buffering and temporary storage of system variables, the area in which faster addressing is of most value. Cycle-by-cycle operation is shown in Table 14 for Extended Addressing.

TABLE 13 – DIRECT MODE CYCLE BY CYCLE OPERATION

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data
CPX LDS LDX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data (High Order Byte)
		4	1	Operand Address + 1	1	Operand Data (Low Order Byte)
STA	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address
		3	0	Destination Address	1	Irrelevant Data (Note 1)
		4	1	Destination Address	0	Data from Accumulator
STS STX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	0	Address of Operand	1	Irrelevant Data (Note 1)
		4	1	Address of Operand	0	Register Data (High Order Byte)
		5	1	Address of Operand + 1	0	Register Data (Low Order Byte)

Note 1. If device which is address during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

FIGURE 31 – EXTENDED ADDRESSING MODE

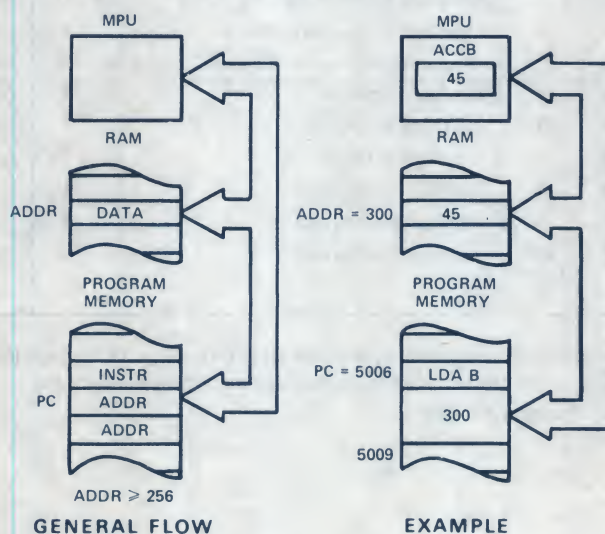


TABLE 14 – EXTENDED MODE CYCLE BY CYCLE

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
STS STX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	0	Address of Operand	1	Irrelevant Data (Note 1)
		5	1	Address of Operand	0	Operand Data (High Order Byte)
		6	1	Address of Operand + 1	0	Operand Data (Low Order Byte)
JSR	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Subroutine (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
		4	1	Subroutine Starting Address	1	Op Code of Next Instruction
		5	1	Stack Pointer	0	Return Address (Low Order Byte)
		6	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		7	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		8	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		9	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
JMP	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Jump Address (High Order Byte)
		3	1	Op Code Address + 2	1	Jump Address (Low Order Byte)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data
CPX LDS LDX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data (High Order Byte)
		5	1	Address of Operand + 1	1	Operand Data (Low Order Byte)
STA A STA B	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address (High Order Byte)
		3	1	Op Code Address + 2	1	Destination Address (Low Order Byte)
		4	0	Operand Destination Address	1	Irrelevant Data (Note 1)
		5	1	Operand Destination Address	0	Data from Accumulator
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Current Operand Data
		5	0	Address of Operand	1	Irrelevant Data (Note 1)
		6	1/0 (Note 2)	Address of Operand	0	New Operand Data (Note 2)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

Note 2. For TST, VMA = 0 and Operand data does not change.



Relative Address Mode — In both the Direct and Extended modes, the address obtained by the MPU is an absolute numerical address. The Relative addressing mode, implemented for the MPU's branch instructions, specifies a memory location relative to the Program Counter's current location. Branch instructions generate two bytes of machine code, one for the instruction opcode and one for the "relative" address (see Figure 32). Since it is desirable to be able to branch in either direction, the 8-bit address byte is interpreted as a signed 7-bit value; the 8th bit of the operand is treated as a sign bit, "0" = plus and "1" = minus. The remaining seven bits represent the numerical value. This results in a relative addressing range of ± 127 with respect to the location of the branch instruction itself. However, the branch range is computed with respect to the next instruction that would be executed if the branch conditions are not satisfied. Since two bytes are generated, the next instruction is located at PC + 2. If D is defined as the address of the branch destination, the range is then:

$$(PC + 2) - 127 \leq D \leq (PC + 2) + 127$$

or
$$PC - 125 \leq D \leq PC + 129$$

that is, the destination of the branch instruction must be within -125 to +129 memory locations of the branch instruction itself. For transferring control beyond this range, the unconditional jump (JMP), jump to subroutine (JSR), and return from subroutine (RTS) are used.

In Figure 32, when the MPU encounters the opcode for BEQ (Branch if result of last instruction was zero), it tests the Zero bit in the Condition Code Register. If that bit is "0", indicating a non-zero result, the MPU continues execution with the next instruction (in location 5010 in Figure 32). If the previous result was zero, the branch condition is satisfied and the MPU adds the offset, 15 in this case, to PC + 2 and branches to location 5025 for the next instruction.

The branch instructions allow the programmer to efficiently direct the MPU to one point or another in the control program depending on the outcome of test results. Since the control program is normally in read-only memory and cannot be changed, the relative address used in execution of branch instructions is a constant numerical value. Cycle-by-cycle operation is shown in Table 15 for relative addressing.

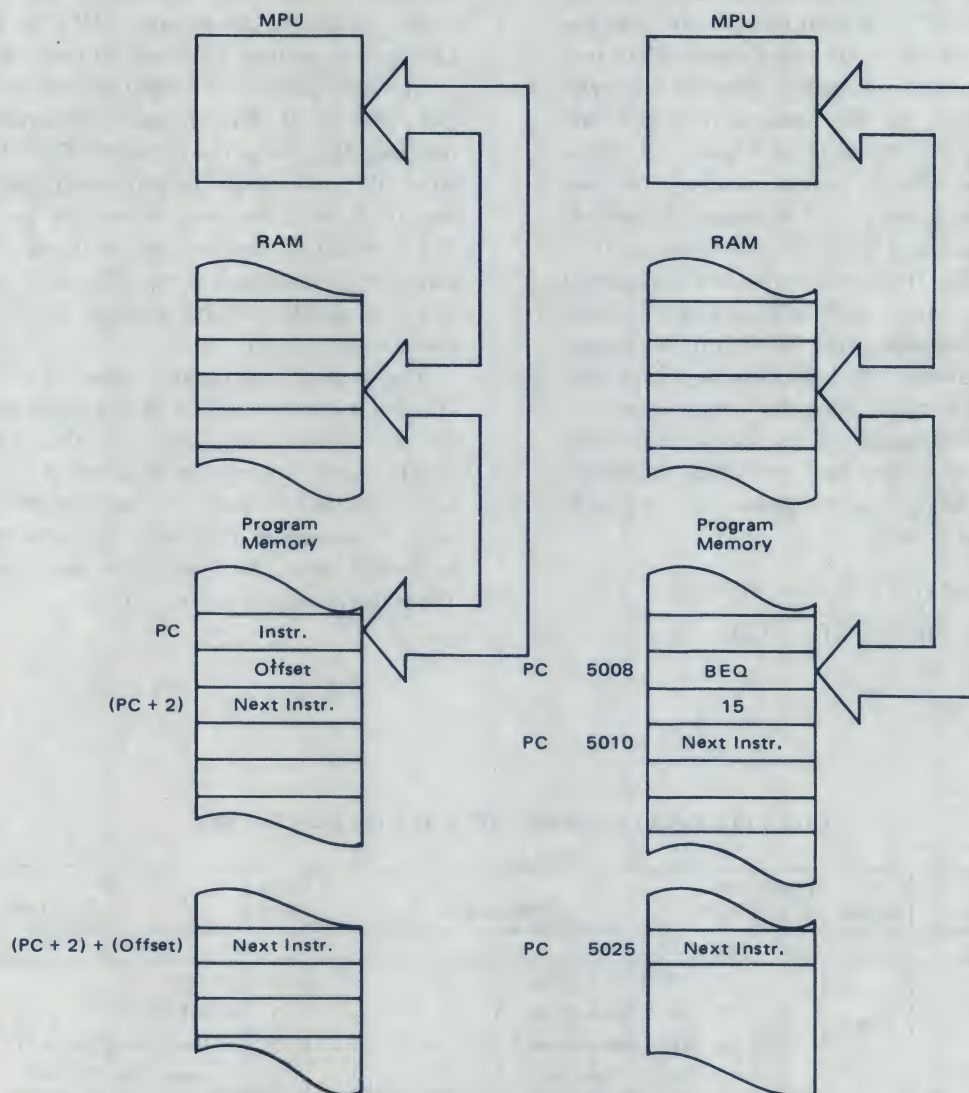
TABLE 15 — RELATIVE MODE CYCLE-BY-CYCLE OPERATION

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
BCC BHI BNE BCS BLE BPL BEQ BLS BRA BGE BLT BVC BGT BMI BVS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		4	0	Branch Address	1	Irrelevant Data (Note 1)
BSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer - 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer - 2	1	Irrelevant Data (Note 1)
		7	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		8	0	Subroutine Address	1	Irrelevant Data (Note 1)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.



FIGURE 32 — RELATIVE ADDRESSING MODE



Indexed Addressing Mode — With Indexed addressing, the numerical address is variable and depend on the current contents of the Index Register. A source statement such as

Operator	Operand	Comment
STAA	X	PUT A IN INDEXED LOCATION

causes the MPU to store the contents of accumulator A in the memory location specified by the contents of the Index Register (recall that the label "X" is reserved to designate the Index Register). Since there are instructions for manipulating X during program execution (LDX, INX, DEX, etc.), the Indexed addressing mode provides a dynamic "on the fly" way to modify program activity.

The operand field can also contain a numerical value that will be automatically added to X during execution. This format is illustrated in Figure 33.

When the MPU encounters the LDAB (Indexed) opcode in location 5006, it looks in the next memory location for the value to be added to X (5 in the example) and calculates the required address by adding 5 to the present Index Register value of 400. In the operand format, the offset may be represented by a label or a numerical value in the range 0-255 as in the example. In the earlier example, STAA X, the operand is equivalent to 0,X, that is, the 0 may be omitted when the desired address is equal to X. Table 16 shows the cycle-by-cycle operation for the Indexed Mode of Addressing.

FIGURE 33 — INDEXED ADDRESSING MODE

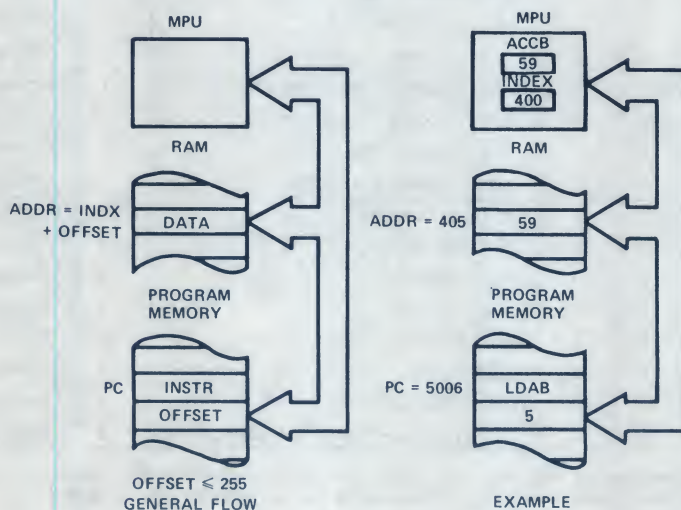


TABLE 16 – INDEXED MODE CYCLE BY CYCLE

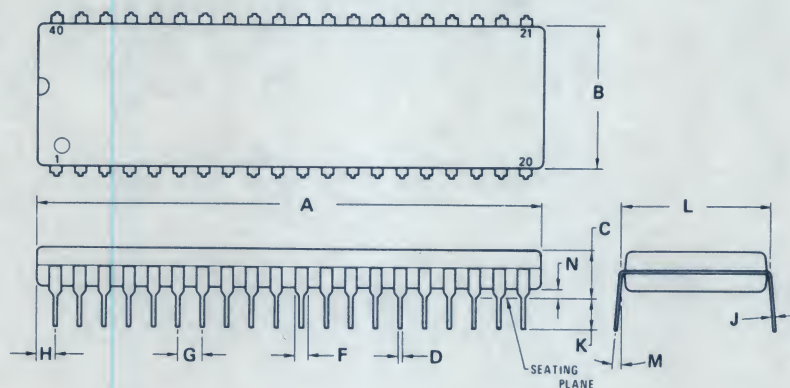
Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
INDEXED						
JMP	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data
CPX LDS LDX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data (High Order Byte)
		6	1	Index Register Plus Offset + 1	1	Operand Data (Low Order Byte)
STA	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Current Operand Data
		6	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		7	1/0 (Note 2)	Index Register Plus Offset	0	New Operand Data (Note 2)
STS STX	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data (High Order Byte)
		7	1	Index Register Plus Offset + 1	0	Operand Data (Low Order Byte)
JSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer - 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer - 2	1	Irrelevant Data (Note 1)
		7	0	Index Register	1	Irrelevant Data (Note 1)
		8	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

Note 2. For TST, VMA = 0 and Operand data does not change.



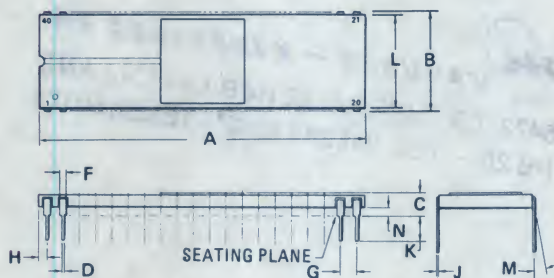
PACKAGE DIMENSIONS



NOTES:

1. LEADS TRUE POSITIONED WITHIN 0.25 mm (0.010) DIA AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION (DIM "D").
2. DIM "L" TO CENTER OF LEADS WHEN FORMED PARALLEL.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	51.82	52.32	2.040	2.060
B	13.72	14.22	0.540	0.560
C	4.57	5.08	0.180	0.200
D	0.36	0.51	0.014	0.020
F	1.02	1.52	0.040	0.060
G	2.54 BSC		0.100 BSC	
H	1.65	2.16	0.065	0.085
J	0.20	0.30	0.008	0.012
K	3.05	3.56	0.120	0.140
L	15.24 BSC		0.600 BSC	
M	0°	10°	0°	10°
N	0.51	1.02	0.020	0.040

CASE 711-02
(PLASTIC)

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	50.29	51.31	1.980	2.020
B	14.86	15.62	0.585	0.615
C	2.54	4.19	0.100	0.165
D	0.38	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
H	0.76	1.78	0.030	0.070
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
L	14.60	15.37	0.575	0.605
M	0°	10°	0°	10°
N	0.51	1.52	0.020	0.060

NOTE:

1. LEADS, TRUE POSITIONED WITHIN 0.25 mm (0.010) DIA (AT SEATING PLANE), AT MAX. MAT'L CONDITION.


CASE 715-02
(CERAMIC)

Circuit diagrams utilizing Motorola products are included as a means of illustrating typical semiconductor applications; consequently, complete information sufficient for construction purposes is not necessarily given. The information has been carefully checked and

is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the semiconductor devices described any license under the patent rights of Motorola Inc. or others.



MOTOROLA Semiconductor Products Inc.

 MANUDAX - NEDERLAND BV
5473 ZG HEESWIJK (N.B.) - HOLLAND
PB 25 — TEL. 04139 - 1252* - TELEX 50175



MOTOROLA Semiconductor Products Inc.

133-E78/10.0

Printed in Switzerland